Essays in computational finance

PhD thesis submitted September 18, 2024.

Author
Johan Auster
johan.auster@math.ku.dk
Department of Mathematical Sciences
Universitetspark 5
2100 Copenhagen Ø, Denmark

Supervisor
Rolf Poulsen
Professor
Department of Mathematical Sciences
University of Copenhagen

Assessment committee (chair)
David Skovmand
Associate Professor
Department of Mathematical Sciences
University of Copenhagen

Assessment committee (external member)
Natalie Packham
Professor
Berlin School of Economics and Law

Assessment committee (external member)
Artur Sepp
Head Quant
LGT Bank AG

ISBN 978-87-7125-234-7

Contents

Abstract (English)

This dissertation contains an assortment of articles pertaining to mathematical finance and computational methods. A total of four articles are included, covering a wide range of topics detailed below.

The first article covers various methods for the approximation of the exponential function with relevant adjustments specific to implementations using *fixed-point numbers*, which are of particular relevance in *decentralized finance* applications, where (the more commonly-used) floating-point numbers are typically not available.

The second article demonstrates the utilization of *dual numbers* in the *diffusion operator integral* variance reduction method proposed by Heath and Platen. Using dual numbers for exact differentiation allows the method to be extended to complex option pricing problems that have existing solutions in the Black-Scholes model without the need for error-prone derivations of analytical sensitivities. The method is applied to the pricing of discretely-monitored down-and-out call barrier options and floating-strike lookback put options.

The third article introduces a generalization of a scalable reward distribution method frequently used in practical implementations of smart contracts. This generalization allows for the constant-time distribution of an arbitrary *claims process* among accounts according to their (changing) relative shares. Special cases covering deposit-backed claims and delayed allocation are covered, along with a formalization of blockchains, smart contracts and associated filtrations that allows for an interpretation consistent with the presented continuous-time setup.

The fourth and final article presents a method for identifying the "most equitable" distribution of an integer budget among integer allocations when constraints apply to individual allocations. In a first step, a table characterization of real-valued solutions is constructed, which can be cached (stored) for future use. A second step then retrieves integer allocations that solve the original problem from this characterization. An example application to the allocation of space in graphical user interfaces is then shown.

Abstract (Danish)

Denne afhandling indeholder en række artikler indenfor matematisk finansiering og computional metoder. Fire artikler er inkluderet og opsummeret nedenfor.

Den første artikel redegører for adskillige metoder til approksimering af eksponential funktionen med justeringer for *fixed-point tal*, hvilket har særlig relevans for anvendelser inden for *decentraliseret finansiering*, hvor (de mere hyppigt anvendte) floating-point tal typisk ikke er tilgængelige.

Den anden artikel demonstrerer brugen af *dual tal* i *diffusions operator integral* varians reduktions metoden introduceret af Heath og Platen. Brugen af dual tal til eksakt differentiering udvider metoden til options pristfastsættelses problemer med eksisterende løsninger i Black-Scholes modellen uden at kræve analytiske løsninger til prisfølsomheder. Metoden er vist i prisfastsættelsen af diskret-overvågede down-and-out call barrier optioner og floating-strike lookback put optioner.

Den tredje artikel introducerer en generalisering af en skalerbar belønning distribuerings metode ofte anvendt i praktiske implementeringer af smart kontrakter. Denne generalisering kan anvendes til distribuering af en viklårlig *claims proces* blandt kontoer i forhold til deres (ændrende) relative andele. Særtilfælde med claims opbakket af depoter og forsinket allokering af claims er beskrevet, samt en formalisering af blockchains, smart kontrakter og associerede filtreringer, hvilket tillader en fortolkning der er forenelig med det præsenterede setup i kontinuert tid.

Den fjerde og sidste artikel præsenterer en metode til identificering af den "mest retfærdige" distribuering af et heltals budget blandt heltal allokeringer med begrænsninger på individuelle allokeringer. En tabel der karakteriserer reale løsninger er konstrueret i et første trin. I et andet trin bliver heltal løsninger afledt fra denne tabel, hvilket løser det oprindelige heltal allokerings problem. Et eksempel med anvendelse inden for allokering af plads i grafiske brugerflader er vist.

Approximation of the exponential function with fixed-point numbers

Johan Auster*

Abstract

This article presents a non-technical overview of popular methods for approximation of the exponential function with notes and adjustments relevant for fixed-point number implementations. A preliminary discussion of fixed-point numbers is included, after which Taylor polynomials, Padé approximations, range reduction with bit-shifting and minimax methods are covered and compared. A practical example of configuring a fixed-point implementation based on accuracy requirements for the value of a deposit after interest rate compounding is then provided, followed by concluding remarks.

Keywords: Approximation theory, exponential function, fixed-point arithmetic, Taylor polynomial, Padé approximation, range reduction, minimax approximation.

*Department of Mathematical Sciences, University of Copenhagen, Denmark. E-mail: johan.auster@math.ku.dk.

1 Introduction

This paper provides a survey of common approximation methods relevant to the implementation of exponential function approximators. The focus will be on usage with fixed-point numbers, with relevant adjustments and discussion of how these implementations would differ from "standard" floating-point implementations.

A balance between brevity and providing sufficient insights for developers to comfortably apply the methods covered is strived for. The aim of the article is to provide a practical perspective with an emphasis on points relevant to smart contract development in particular; hence the focus on fixed-point numbers and count of operations (due to the per-operation transaction costs on blockchains).

Approximation theory is a deep topic with rich and rigorous underlying results, remaining an active area of research even after centuries of developments. Given the priority on practice and the amount of methods covered, the formal theory and results are, broadly speaking, left out. For a more extensive theoretical treatment of general approximation theory, we refer to textbooks such as the one by Powell (1981).

Section 2 provides a preliminary introduction to fixed-point arithmetic and how these representations can be used to emulate non-integer numbers in purely-integer computational environments. Section 3 compares multiple methods commonly used in practice for approximation of the exponential functions, starting with Taylor polynomials, rational Padé approximations, range reduction and finally minimax methods. Section 4 demonstrates the calibration of a fixed-point approximator based on practical requirements. Section 5 provides concluding remarks.

2 Fixed-point numbers

This section provides a brief preliminary discussion of *fixed-point numbers* and their properties, as the focus of the article will be on implementation for this representation of numbers.[1] Beyond this section, fixed-point operations will be assumed implicitly (e.g. in appropriate rescaling and floored division), though points regarding these specifics as they relate to the implementation of a given approximation method under consideration will still be discussed. More details on fixed-point arithmetic can be found in e.g. the report by Yates (2020) and all fixed-point implementations used in this article can be found in the open-sourced `github` repository referenced in Section 3.1.

A fixed-point number is a digital representation of a real number x consisting of an internal integer representation $\bar{x} \in \mathbb{N}$ representing counts of units of some fixed scaling factor $S \in \mathbb{R}$. For example, the integer $\bar{x} = 51$ with scaling factor $S = 1/2\pi$ would represent the number $\bar{x} \cdot S = 51/2\pi$.

One of the most common configurations of fixed-point numbers is base-10 scaling:

$$S_{10}(d) \coloneqq \frac{1}{10^d}, \quad d \in \mathbb{Z} \setminus \{0\}, \tag{2.1}$$

which represents decimal numbers for a fixed number of digits $d > 0$, or alternatively: truncation of integer values with $d < 0$.

[1] While similar in name, fixed-point numbers is unrelated to the concept of *fixed points* (of e.g. a function), which refers to points that are unchanged after a transformation.

Another common configuration is binary scaling:

$$S_2(d) \coloneqq \frac{1}{2^d}, \quad d \in \mathbb{Z} \setminus \{0\}, \tag{2.2}$$

which is a natural choice given the binary representation of integers in computing. For the remainder of the article, we always assume base-10 scaling (2.1) with $d > 0$.

In practice, scaling factors are typically treated as implicit, with the expected scaling factor of inputs being documented in source code where relevant. Fixed-point numbers are thereby not reliant on anything other than the availability of integers, since their interpretation as units of a given scaling factor does not require an explicit inclusion in the underlying digital representation.

For any two fixed-point numbers x_1, x_2 with scaling factors S_1, S_2, one can convert x_1 to the same representation as x_2 by multiplication of the (floored) ratio of scaling factors $\lfloor {}^{(\bar{x}_1 \cdot S_2)}/_{S_1} \rfloor$, which may be subject to rounding and loss of precision depending on the scaling factors involved.

Assuming the same scaling factor S for two fixed-point numbers x_1, x_2, addition and subtraction is straight-forward: the operations can be directly applied to the internal integers and the result remains a fixed-point number of the same scaling factor. Multiplication- and division by non-fixed-point integers similarly requires no rescaling,[2] though integer division may be subject to truncation errors when the numerator is not divisible by the divisor.

Multiplication and division between two fixed-point numbers ($S \neq 1$) however leads to scaling issues, since multiplying e.g. $\bar{x}_1 = 10$ and $\bar{x}_2 = 50$ both with scaling factor $S = {}^1/_{10}$ yields $\bar{x}_1 \cdot \bar{x}_2 = 500$, which would represent 50.0 rather than 5.0. More generally, the product of fixed-point numbers will be in the form of a fixed-point number with scaling factor equal to the product of the scaling factors of the multiplicants, thereby requiring rescaling consistent with the desired representation, e.g. $\lfloor {}^{(500 \cdot S^2)}/_{S} \rfloor = \lfloor {}^{500}/_{10} \rfloor = 50$.

Conversely, "naive" division of two fixed-point numbers can lead to severe truncation error due to the implicit flooring of integer division, as the result of $\lfloor {}^{n_1}/_{n_2} \rfloor$ will have a scaling factor of $\lfloor {}^{S_1}/_{S_2} \rfloor$. In practice one often circumvents this by preemptively rescaling the numerator to $S_1 \cdot S_2$ representation, which leads to a result with scaling factor S_1 of the numerator before rescaling.

Fixed-point numbers are especially suited for domains with known units and ranges of values, where scaling factors can be chosen according to the precision required in the relevant context. For example, accounting software configured for USD may make use of a decimal scaling factor $S = {}^1/_{100}$ such that $\bar{x} \in \mathbb{N}_0$ will represent dollar amounts in cents. In contrast, the more widely-used representation of real-valued numbers in modern computing are floating-point numbers, which represent numbers as multiples m of a fixed base b raised to some (variable) exponent c, similar to scientific notation.

Advantages of fixed-point numbers include their relative simplicity in comparison to floating-point numbers, low computational cost of operations and ability to exactly represent (integer counts) of real numbers, thereby not being subject to the same error propagation issues as occur in floating-point arithmetic. However, these advantages come at

[2] Equivalently one can think of "non-fixed-point integers" as fixed-point integers with $S = 1$, in which case the rescaling covered in the next paragraph corresponds to division by 1 and can thus be skipped.

the cost of some flexibility, as the appropriate scaling factor can vary wildly depending on the context and carries with it a trade-off between precision in the representation and the range of numbers that can be represented within the number of bits available.

While fixed-point numbers are still utilized in certain low-compute environments (such as field-programmable gate arrays) and accounting (for exact representation of dollar amounts), floating-point numbers are the norm in modern general-purpose computing. With the popularization of blockchains and decentralized finance applications however, the domain of fixed-point numbers has expanded, as e.g. computation of continuous interest rate compounding requires (high-precision) fixed-number approximation algorithms that internally utilize purely-integer representations of currency amounts.

The high-bit integer environment with serial execution of instructions across multiple validator nodes in blockchain Virtual Machines (VMs) stands in stark contrast to the computational methods often seen in e.g. machine learning applications, which generally makes use of highly-parallel low-bit floating-point computations executed on Graphics Processing Units (GPUs) or even more specialized hardware. Methods well-suited for one computational environment may be poorly suited for the other.

With the expensive nature of on-chain operations,[3] there is a renewed importance of developer familiarity with low-operation approximation methods that are often taken for granted in the standard libraries of modern programming languages. Although many approximation methods have straight-forward analogues for fixed-point numbers (as one can simply appropriately rescale after every operation that requires it), the fixed-point paradigm can lead to different formulations, or even entire methods, being more suitable than the approaches used for the same problems in floating-point arithmetic.[4]

Even with thoroughly tested implementations available, the nature of fixed-point numbers puts far greater responsibility on developers to configure the precision of the fixed-number representations in line with expected input ranges to ensure stable and valid results for the specific context. Implementing and validating such numerical approximation algorithms and configuring these appropriately can be a large undertaking for developers coming from modern general-purpose languages without previous experience in numerical methods.

The above partially motivates the subsequent coverage of approximation methods that are relatively straight-forward to implement across any architecture that supports basic integer operations. After discussing common methods for approximation and adjustments for fixed-point implementations in Section 3, a practical example of configuring a fixed-point approximation method is given in Section 4.

[3] Relative to equivalent general-purpose off-chain computation.

[4] As one example of this point, modern computational architectures often make heavy use of Single Instruction, Multiple Data (SIMD) instructions that execute multiple operations in a single instruction, making an assessment of efficiency based solely on a simple count of operations somewhat misleading. In contrast, blockchain VMs tend to utilize simple serial instruction sets for arithmetic with fixed transaction costs assigned on a per-operation basis.

3 Approximation of the exponential function

3.1 Implementation details

All approximators presented in the following sections are implemented in `Python 3.12` for fixed-point integer inputs and utilize $d = 20$ digits unless otherwise stated. For estimation of relative errors, approximations are compared against the output of the exponential function provided in the arbitrary-precision `mpmath` library targeting the decimal precision matching the number of digits in the fixed-point representation.

Integer division is assumed to correspond to *floored* division as suggested by Knuth (1997), which matches the implementation in `Python`, the `Ethereum Virtual Machine` (`EVM`) (Buterin, 2013) and many other programming environments. Adjustments may be required when implementing the methods outlined in the following sections in environments which do not use floored division by default.

Variations or reformulations motivated by the fixed-point setting are made note of. Otherwise, the implementation specifics of fixed-point numbers are treated as implicit, including e.g. rescaling of constants and rescaling before division / after multiplication between two fixed-point numbers, as outlined in Section 2.

For any further details on the implementation, the full source code is made available online on `https://github.com/austerj/exponential-approximation`, which includes a full set of unit tests and reproduction of all figures and results in the article.

3.2 Taylor polynomial approximation

The *Taylor series* (Taylor, 1717) of an infinitely differentiable function $f : \mathbf{U} \to \mathbb{R}$ for an open subset $\mathbf{U} \subseteq \mathbb{R}$ is the well-known power series expansion

$$\sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i \tag{3.1}$$

for $x, a \in \mathbf{U}$ with $f^{(i)}$ denoting the i'th derivative of f. When f is equal to its Taylor series, it is said to be *analytic*. This characterization of a sufficiently regular function as an infinite sum of terms that depend only on evaluations of the derivatives of f, a factorial $i!$ and an integer power of $x - a$ is a fundamental result in analysis.

One is rarely able to evaluate (3.1) in full;[5] fortunately, Taylor's theorem (Kudryavtsev, 2013) shows that summation of a *finite* number $N \in \mathbb{N}$ of the terms in (3.1), known as the *Taylor polynomial* of f:

$$P_N(x) \coloneqq \sum_{i=0}^{N} \frac{f^{(i)}(a)}{i!}(x-a)^i \tag{3.2}$$

acts an approximator of the original function f with the remainder term

$$f(x) - P_N(x) = o\left(|x-a|^N\right) \tag{3.3}$$

[5]The notable exception being when f is itself a polynomial, in which case the derivatives eventually become constants equal zero on the full domain, resulting in a finite number of non-zero terms.

as $a \to x$, i.e. the approximation error of the order-N Taylor polynomial of f is bounded by $|x-a|^N$ in the neighbourhood of a.

While (3.3) rapidly goes to zero as x approaches a (in particular for large N), this says nothing about the errors outside the neighbourhood of a. This already hints at one potential shortcoming of Taylor polynomials, namely that their accuracy may rapidly deteriorate when used for extrapolation at points x **not** in the immediate vicinity of a.

Applying the expansion (3.1) to e^x around $a = 0$ yields the remarkably simple expression

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \tag{3.4}$$

since $(e^x)^{(i)}(a) = e^0 = 1$ for all $i \in \{0, 1, \ldots\}$.

When implementing the corresponding order-N Taylor polynomial from (3.4), one can use that

$$\sum_{i=0}^{N} \frac{x^i}{i!} = \sum_{i=0}^{N} \frac{N!}{N!} \frac{x^i}{i!} = \frac{\sum_{i=0}^{N} \frac{N!}{i!} x^i}{N!} \tag{3.5}$$

and accumulate the numerator via Horner's scheme (Horner, 1819):

$$\sum_{i=0}^{N} \frac{N!}{i!} x^i = \frac{N!}{0!} + x\left(\frac{N!}{1!} + x\left(\frac{N!}{2!} + \ldots + x\left(\frac{N!}{(N-1)!} + \frac{N!}{N!}x\right)\right)\right) \tag{3.6}$$

followed by a single division by $N!$ to correct for the integer multiplications.

While Horner's scheme is optimal in the sense of minimizing multiplications and additions in the evaluation of a general polynomial (Pan, 1966), the rearrangement in (3.5) does come at the cost of one additional subsequent division. This reformulation however has the advantage of avoiding the propagation of errors in fixed-point numbers by circumventing the $i!$ integer division in each term, replacing these with integer *multiplications* (which do **not** result in loss of precision).[6]

The relative errors of Taylor approximations are shown in Figure 1. As suggested by the previous discussion, errors rapidly increase as x moves away from $a = 0$, with odd-ordered approximations even going negative for sufficiently small values of x.

Despite the high regularity of the exponential function, the failure of polynomials to maintain a high degree of accuracy outside the neighbourhood of a is not at all surprising upon reflection of the type of function we are approximating: namely, we are trying to replicate a function with exponential growth / decay by polynomials of finite degrees. This is bound to eventually fail as x moves sufficiently far away from a in either direction, as exponential growth will *always* overtake that of **any** polynomial - and likewise for exponential decay, as polynomials will always diverge to $\pm\infty$ as $x \to -\infty$.

[6]Since the scheme evaluates polynomials by repeated multiplications, any such truncation errors will propagate and grow exponentially, although their magnitude will depend on both the orders used in the Taylor polynomial and the number of digits in the fixed-point representation.

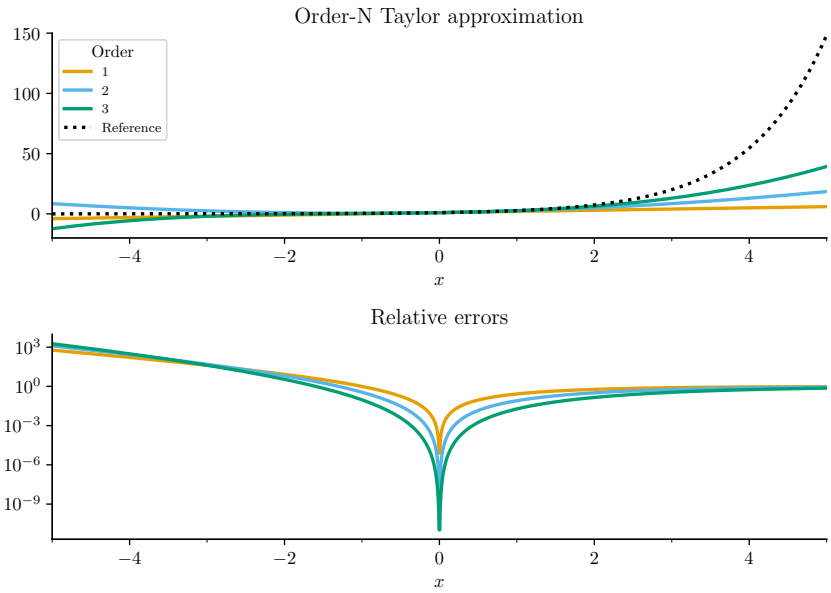


Figure 1: Relative errors of order-N Taylor approximations.

Despite predating the first digital computer by centuries, Taylor polynomials are often the starting point of extending the (typically hardware-level) arithmetic operations provided by a general-purpose computer to the numerical evaluation of a wide range of mathematical functions, such as trigonometric functions.

In practice, one can often find improvements on Taylor polynomials, whether the aim is to minimize errors within an interval, replicate certain asymptotic behavior of the target function (e.g. cyclical functions or functions that converge to a constant), achieve similar accuracy with fewer operations or some combination of all of these goals. Nevertheless, Taylor polynomials are often a solid foundation to build upon, both in its underlying theory and as a starting point to guide the choice of a more advanced methods by identifying which shortcomings are most relevant to a given problem.

One category of methods that often provides substantial improvements are approximators which make use of a *ratio* of polynomials. Padé approximation is one such method which, similar to Taylor polynomials, replicates the behavior of a function in a neighbourhood of a point; only with a ratio of polynomials of potentially different degrees. The subsequent section covers this approach further for the exponential function with direct comparison to Taylor polynomials.

3.3 Padé rational polynomial approximation

The order-$[N/M]$ *Padé approximation* (Padé, 1892) with $N \in \mathbb{N}$ and $M \in \mathbb{N}_0$ of a $N+M$-times differentiable function $f : \mathbf{U} \to \mathbb{R}$ for an open subset $\mathbf{U} \subseteq \mathbb{R}$ is the *ratio* of polynomials

$$R_{[N/M]}(x) := \frac{\sum_{i=0}^{N} c_i^N x^i}{\sum_{i=0}^{M} c_i^M x^i} \tag{3.7}$$

with real-valued coefficients $c_i^N, c_i^M \in \mathbb{R}$, which for some fixed $a \in \mathbf{U}$ satisfies

$$f^{(i)}(a) = R^{(i)}(a) \tag{3.8}$$

for all $i \in \{0, 1, \ldots, N+M\}$, i.e. the Padé approximation is a ratio of polynomials whose value and first $N+M$ derivatives are consistent with those of f at the point a.

When f is transcendental, the coefficients of (3.7) can be found from solving a system of linear equations (Weisstein, 2023), though algorithms exist to compute the coefficients of (3.7) in more general cases (Wynn, 1966). Padé approximations are unique when they exist and can be seen as a generalization of Taylor polynomials (3.2), since the order-$[N/0]$ Padé approximation will be identical to the order-N Taylor polynomial.

A key difference from Taylor polynomials in the behavior of the rational structure is the potential presence of *poles*, i.e. points at which the denominator in (3.7) are zero. The rational specification also allows for approximations that converge to constants, which is not possible with a polynomial. In this sense, Padé approximations are "more flexible" than Taylor polynomials.

While the Padé coefficients are chosen to replicate the features of f only around a singular point, multi-point extensions of the method exist to find rational functions in the form of (3.7) that replicate the features discussed above (i.e. poles or specific values) of f at fixed points or asymptotically (Ismail, 1996). Even still, the choice of N and M provide additional control over the global behavior of the approximator and, unlike Taylor polynomials, rational approximations do not necessarily diverge as $|x| \to \infty$.

The Padé coefficients satisfying (3.8) around $a = 0$ for the exponential function have the analytical solution (Ehle, 1969):

$$c_i^N = \frac{(N+M-i)!N!}{(N+M)!i!(N-i)!}, \quad c_i^M = \frac{(N+M-i)!M!}{(N+M)!i!(M-i)!}(-1)^i \tag{3.9}$$

and taking $N = M$, we get

$$\begin{aligned} R_{[N/N]}(x) &= \frac{\sum_{i=0}^{N} \frac{(2N-i)!N!}{(2N)!i!(N-i)!}x^i}{\sum_{i=0}^{N} \frac{(2N-i)!N!}{(2N)!i!(N-i)!}(-x)^i} = \frac{\frac{(2N)!}{N!}}{\frac{(2N)!}{N!}} \frac{\sum_{i=0}^{N} \frac{(2N-i)!N!}{(2N)!i!(N-i)!}x^i}{\sum_{i=0}^{N} \frac{(2N-i)!N!}{(2N)!i!(N-i)!}(-x)^i} \\ &= \frac{\sum_{i=0}^{N} \frac{(2N-i)!}{i!(N-i)!}x^i}{\sum_{i=0}^{N} \frac{(2N-i)!}{i!(N-i)!}(-x)^i} = \frac{A(x)+B(x)}{A(x)-B(x)} \end{aligned} \tag{3.10}$$

with even- and odd term sums

$$A \coloneqq \sum_{i=0}^{N} \mathbf{1}_{\{0,2,\ldots\}}(i) \frac{(2N-i)!}{i!(N-i)!}x^i, \quad B \coloneqq \sum_{i=0}^{N} \mathbf{1}_{\{1,3,\ldots\}}(i) \frac{(2N-i)!}{i!(N-i)!}x^i. \tag{3.11}$$

Rescaling the coefficients in (3.10) is particularly advantageous for fixed-point representations as this leads to integer coefficients, thereby avoiding the need for rescaling after multiplying powers of x by a non-integer (i.e. fixed-point) coefficient. This also saves a coefficient multiplication, since the highest-order term coefficient

$$\frac{(2N-N)!}{N!(N-N)!} = \frac{N!}{N!} = 1, \tag{3.12}$$

while the lower-order term is constant since $x^0 = 1$, thus requiring no multiplication. In the non-rescaled formulation, this multiplicative identity is "wasted" on the constant term, which does not require a multiplication in any case.

An additional convenient property for (unsigned) fixed-point numbers is that

$$R_{[N/N]}(-x) = \frac{1}{R_{[N/N]}(x)} = \frac{A(x) - B(x)}{A(x) + B(x)}, \tag{3.13}$$

leading to a straight-forward implementation for the computation of the inverse exponential function without relying on negative integers or having to invert an approximation for a positive input, which may require very large intermediary values to maintain accuracy for high-digit representations due to rescaling, as discussed in Section 2.

Since we have different polynomials in the numerator and denominator, we can no longer rely on a single evaluation of Horner's scheme as with the Taylor polynomial. Instead, the sum computations (3.11) can be implemented with an additional variable for storing computed values of powers of x, which can then be reused for the subsequent coefficient multiplication and addition to even / odd sum accumulators initialized to zero.[7]

Focusing on the $N = M$ case may seem unmotivated at first sight, however there are several benefits to this choice that, especially when working with fixed-point numbers, render this the ideal choice in almost all circumstances:

1. The property in (3.12) no longer applies to both the numerator- and denominator coefficients if $N \neq M$, hence an additional multiplication is required.
2. Sharing coefficients saves multiplications, since the same computation of the product of the coefficient and x^i can be reused.
3. As a continuation of the above, the separation into even- and odd term sums in (3.11) means that any additional order term only needs to be added once (to either A or B). This also makes underflow less likely for unsigned integers, since subtraction only happens after A and B have been computed.
4. The multiplication of a d-digit fixed-point number x with e.g. itself ($x^2 = {}^{(\bar{x}\cdot\bar{x})}/_{10^d}$) requires an additional division by 10^d to rescale the result back to a d-digit representation. Even this point alone makes it relatively more "expensive" to go from $[N/N]$ to $[N/N+1]$ than to go from $[N/N+1]$ to $[N+1/N+1]$, since the additional power term is then reused in the numerator.[8]

For a concrete example of the advantages of $N = M$ configurations, consider the following approximations:

[7] For further details on the implementation, see the source code available in the `github` repository linked in Section 3.1

[8] This same point applies to the equivalent-order Taylor polynomial, making these similar in number of operations required for evaluation to the corresponding $[N/N]$ Padé approximation.

$$
\begin{aligned}
R_{[2/2]}(x) &= \frac{12 + 6x + x^2}{12 - 6x + x^2}, \\
R_{[2/3]}(x) &= \frac{60 + 24x + 3x^2}{60 - 36x + 9x^2 - x^3}, \\
R_{[3/3]}(x) &= \frac{120 + 60x + 12x^2 + x^3}{120 - 60x + 12x^2 - x^3}, \\
R_{[3/4]}(x) &= \frac{840 + 360x + 60x^2 + 4x^3}{840 - 480x + 120x^2 - 16x^3 + x^4}, \\
R_{[4/4]}(x) &= \frac{1680 + 840x + 180x^2 + 20x^3 + x^4}{1680 - 840x + 180x^2 - 20x^3 + x^4}.
\end{aligned}
\tag{3.14}
$$

Table 1 summarizes the arithmetic operations required for the computation of each of the approximations in (3.14) for fixed-point numbers when making use of (3.12) and the separation of terms (3.11) in the $[N/N]$ approximations.

Notably, the [2/3] approximation is **more** computationally costly than the [3/3] approximation, requiring one more subtraction and two more multiplications. Even more stark is the difference between the [3/4] and [4/4] approximation, with the [3/4] approximation requiring one more addition, one more subtraction and three more multiplications.

	$+$	$-$	$\cdot$	$/$
$R_{[2/2]}$	2	1	3	2
$R_{[2/3]}$	3	2	7	3
$R_{[3/3]}$	3	1	5	3
$R_{[3/4]}$	5	2	10	4
$R_{[4/4]}$	4	1	7	4

Table 1: Number of operations in fixed-point Padé approximations assuming reuse of powers of x after rescaling.

The relative errors of Padé approximations are shown in Figure 2. Unlike the Taylor polynomials in Figure 1, Padé approximations are no longer bound to diverge in both directions thanks to the "cancelling" nature of having the same order of terms in both the numerator and denominator. It is however apparent that the approximations remain local in nature as relative errors still grow quickly, even overtaking those of Taylor polynomials to become arbitrarily large near the critical points of odd-ordered approximations (where the negative highest-power term in the denominator eventually outgrows the positive terms).

What is perhaps surprising about the Padé approximations is the extent to which they achieve improvements in accuracy of several orders of magnitude near zero compared to Taylor approximations of much higher order. As can be seen in Figure 3 visualizing the maximal relative errors for $|x| \leq {}^{(\log 2)}/_2$ across order-N Taylor approximations and order-$[N/N]$ Padé approximations, achieving comparable accuracies with Taylor polynomials requires such high orders that there is little reason to prefer Taylor polynomials over Padé approximations for the exponential function for any given target accuracy.

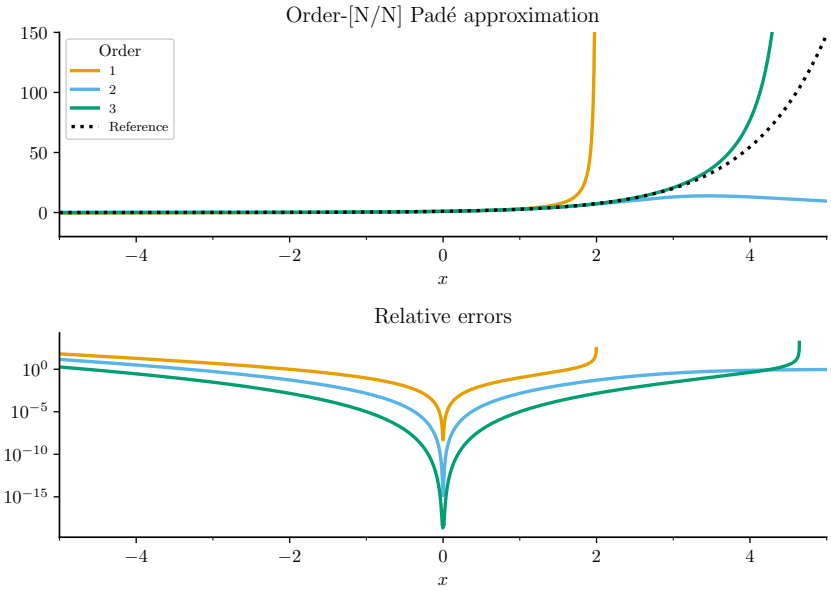


Figure 2: Relative errors of order-$[N/N]$ Padé approximations.

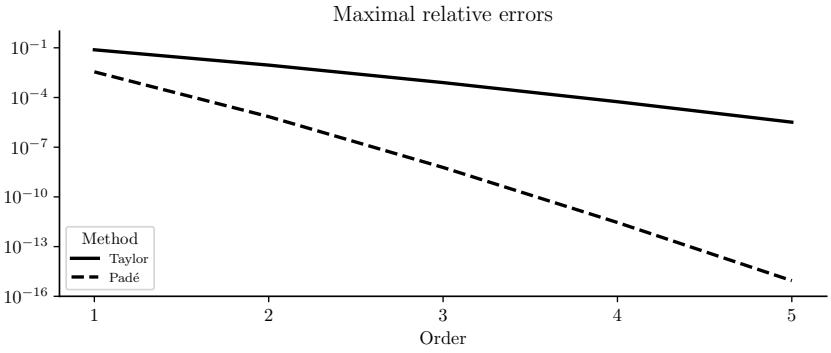


Figure 3: Maximal relative errors for $|x| \leq {}^{(\log 2)}/_{2}$ across order-N Taylor polynomials and order-$[N/N]$ Padé approximations.

From the above discussion, the case for order-$[N/N]$ Padé approximations over Taylor polynomials and $N \neq M$ Padé approximations should be clear, at the very least for local approximations near zero. However, even with the improvements in accuracy, the approximations still deteriorate rapidly to the point of being **practically unusable** once x is sufficiently far from zero. This problem is addressed in the next section via a product decomposition of the exponential function that replicates the relative errors of a small region around zero across the entire domain of real-valued inputs.

3.4 Range reduction and bit-shifting

Having established methods for constructing locally-performant approximators in previous sections, we now exploit properties of the exponential function to achieve accurate approximations across the entire domain. This method is used in e.g. the implementation of the exponential function in the `Cephes` library (Moshier, 1992), and while the earliest reference to the method (known to the author) appears in an article by Cody and Ralston (1967) attributing it to Maehly (1960), no digital version of the latter work appears to be available.

We first note the identity

$$e^x = 2^k 2^{-k} e^x = 2^k e^{\log(2^{-k})} e^x = 2^k e^{x - k\log 2} = 2^k e^{r(x,k)} \tag{3.15}$$

holds for all $x \in \mathbb{R}$ and $k \in \mathbb{Z}$ with remainder

$$r(x,k) \coloneqq x - k\log 2. \tag{3.16}$$

In particular, we can choose

$$k = \left\lfloor \frac{x}{\log 2} + \frac{1}{2} \right\rfloor = \left\lfloor \frac{x + \frac{1}{2}\log 2}{\log 2} \right\rfloor, \tag{3.17}$$

the latter identity being useful to avoid rescaling of fixed-point numbers, such that

$$\begin{aligned} |r(x,k)| &= \left| x - \left\lfloor \frac{x + \frac{1}{2}\log 2}{\log 2} \right\rfloor \cdot \log 2 \right| \\ &= \left| \left(\left(x + \frac{\log 2}{2} \right) \bmod \log 2 \right) - \frac{\log 2}{2} \right| \\ &\leq \max\left(\left\{ \left| \log 2 - \frac{\log 2}{2} \right|, \left| 0 - \frac{\log 2}{2} \right| \right\} \right) = \frac{\log 2}{2} \end{aligned} \tag{3.18}$$

using $0 \leq x \bmod y = x - y\left\lfloor x/y \right\rfloor \leq y$ for $y > 0$.

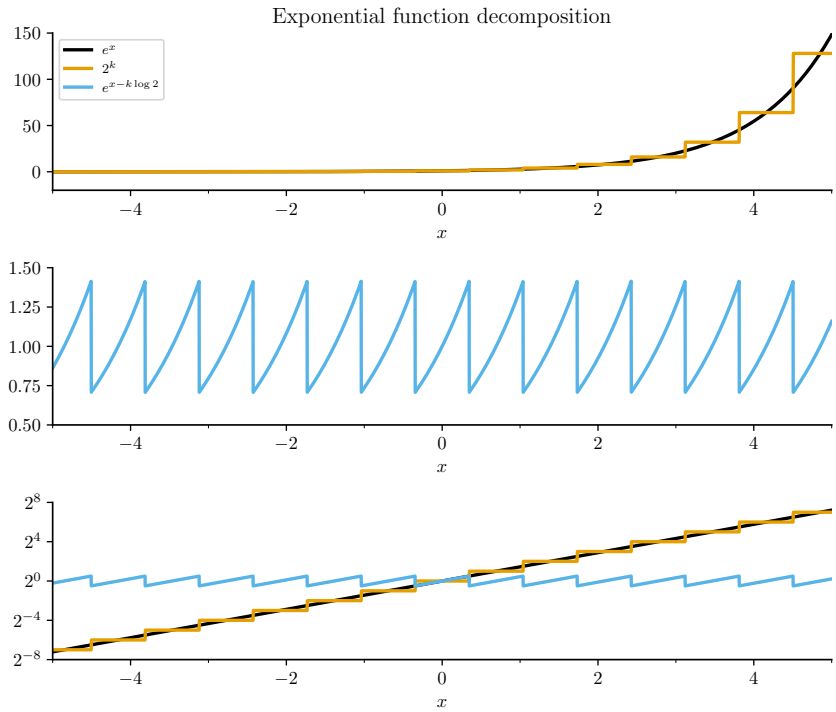


Figure 4: Decomposition $e^x = 2^k e^{x-k\log 2}$ with $k = \left\lfloor \frac{x}{\log 2} + \frac{1}{2} \right\rfloor$.

Figure 4 shows the product decomposition (3.15) visually with the choice of k given by (3.17). On segments of length $\log 2$, the 2^k term acts as a constant, while $e^{r(x,k)}$ is cyclical across these segments, acting as a corrective factor shifting the 2^k constant to the value of e^x. In particular, $e^{r(x,k)} > 1$ when $2^k < e^x$, $e^{r(x,k)} = 1$ when $2^k = e^x$ and $e^{r(x,k)} < 1$ when $2^k > e^x$.

The multiplication by $2^k, k \in \mathbb{Z}$ can be done in a very efficient manner because of the binary representation of integers digitally. Assuming sufficient bits are available for the result, "shifting" the bits of the fixed-point integer k times to the left (to multiply by 2^k) or right (to divide by 2^k) results in the same output as the more computationally expensive equialent of computing 2^k followed by a general multiplication / division operation.

The above enable us to approximate e^x in the following three steps:

1. Find k via (3.17) such that $r(x, k)$ satisfies (3.18).
2. Compute an approximation $y \approx e^{r(x,k)}$.
3. Apply k bit-shifts to y to get $y \cdot 2^k \approx e^x$ via (3.15).

This *range reduction* method effectively restricts the domain of the original exponential function approximator to a reduced domain centered around zero (where the accuracy is relatively good). Bit-shifting is then utilized to efficiently transform the reduced-range approximation to an approximation of the original input by emulating a multiplication or division by 2^k. None of these steps make use of any specific method of approximating $e^{r(x,k)}$, hence this same procedure can be applied to any approximator of the exponential function. Note however that the range reduction method may lead to discontinuities at points $k/2 \log 2, k \in \mathbb{Z}$.

In addition to the increase in accuracy, we avoid the dependency on potentially *very* large intermediary values for high-order approximations that could cause overflow - further extending the domain in which e^x can be approximated in practice. Additionally, the critical points of Padé approximators are no longer an issue, as even the critical point of 2 for the order-[1/1] Padé approximator far exceeds the upper bound $1/2 \log 2 \approx 0.35$ of the reduced domain.

As discussed in Section 3.3, the Padé approximations have considerably lower relative errors near zero compared to Taylor approximations of the same order, though this does not necessarily hold when $|x|$ is large; in particular, Padé approximations may even be undefined. When combined with range reduction however, the high local accuracy of Padé approximations is replicated across the entire domain, as even the order-[1/1] approximations achieve relative errors below 0.4% on the reduced domain. An extension of the range reduction method is demonstrated by Tang (1990) in which the domain is divided further, though this relies on tabulation and several more operations.

Finally, range reduction makes an entire new category of approximators relevant: since we only need a way to approximate e^x on a bounded interval, this opens up the possibilities of extending interpolation-type techniques to work across the entire domain of real-valued inputs. This could include linear interpolation, polynomial interpolation, splines or minimax methods to minimize the maximal absolute error on the bounded interval. The latter example of minimax polynomial- and rational approximations is covered in the next section.

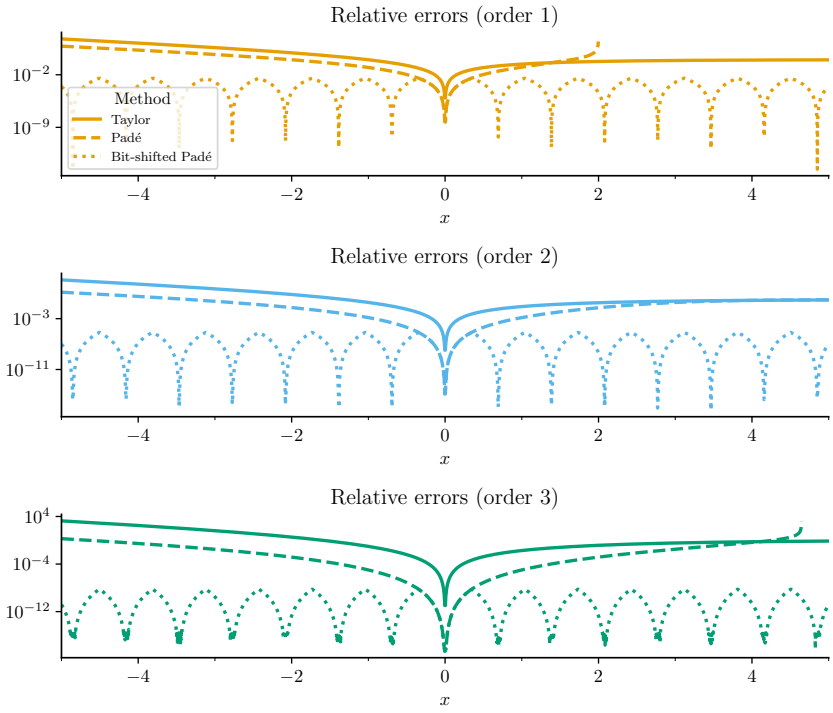


Figure 5: Relative errors of Taylor-, Padé-, and bit-shifted Padé approximations.

3.5 Minimax approximations

Minimax approximators refers to a category of functions which *minimize* the *maximal* **absolute** error compared to a target function f within a bounded interval $[a, b]$, i.e. minimizing the *uniform norm* $L_{\infty,[a,b]}$ of the difference between the approximator and the target function f - in contrast to the approximators discussed in previous sections, which were determined entirely from satisfying conditions at a single point.

In general, fitting a minimax polynomial- or rational function relies on numerical procedures for estimating coefficients. A popular approach to this end is the *Remez* algorithm (Remez, 1934) and extensions thereof, such as as those implemented in the `chebfun` `MATLAB` package for polynomials (Pachon and Trefethen, 2009) and rational approximations (Filip et al., 2018).

While minimax methods are very common in floating-point implementations of non-arithmetic functions and included here for completeness, we will address some caveats in the special case of fixed-point numbers that would support favoring the simpler Padé methods covered in Section 3.3. The theoretical foundations of minimax methods are extensive and covered in many textbooks on approximation theory (e.g. Powell, 1981), however the practical points made in this section will not depend on how the coefficients are obtained, so we simply "take these for granted".[9]

Ignoring the finer details of how one arrives at suitable coefficients, we consider the order-N *minimax polynomial* approximations (analogous to Taylor polynomials):

[9]Coefficients are computed via the `chebfun` package in `MATLAB` - the script is available in the `github` repository linked in Section 3.1. Since these values will be hardcoded constants in practice where the order is fixed, there is no reliance on anything besides the values themselves in "real" implementations.

$$\hat{P}_N(x) \coloneqq \sum_{i=0}^{N} c_i x^i \tag{3.19}$$

with coefficients $c_i \in \mathbb{R}$ that minimize (up to some small tolerance)

$$\left\| \hat{P}_N(x) - f(x) \right\|_{\infty,[a,b]} = \max_{a \le x \le b} \left| \hat{P}_N(x) - f(x) \right|. \tag{3.20}$$

Similarly, we have order-$[N/M]$ *minimax rational* (analogous to Padé) approximations:

$$\hat{R}_{[N/M]}(x) \coloneqq \frac{\sum_{i=0}^{N} c_i^N x^i}{\sum_{i=0}^{M} c_i^M x^i} \tag{3.21}$$

with coefficients $c_i^N, c_i^M \in \mathbb{R}$ that minimize the uniform norm analogous to (3.20). For simplicity, we focus on the $N = M$ case.

Figure 6 compares the relative errors of Taylor polynomials with those of minimax polynomials of the same order (on the left side), as well as Padé approximations with the minimax rational approximation of the same order (on the right side).[10] The minimax coefficients are fitted on the range-reduced region $[a, b] = [-1/2 \log 2, 1/2 \log 2]$ as covered in Section 3.4.

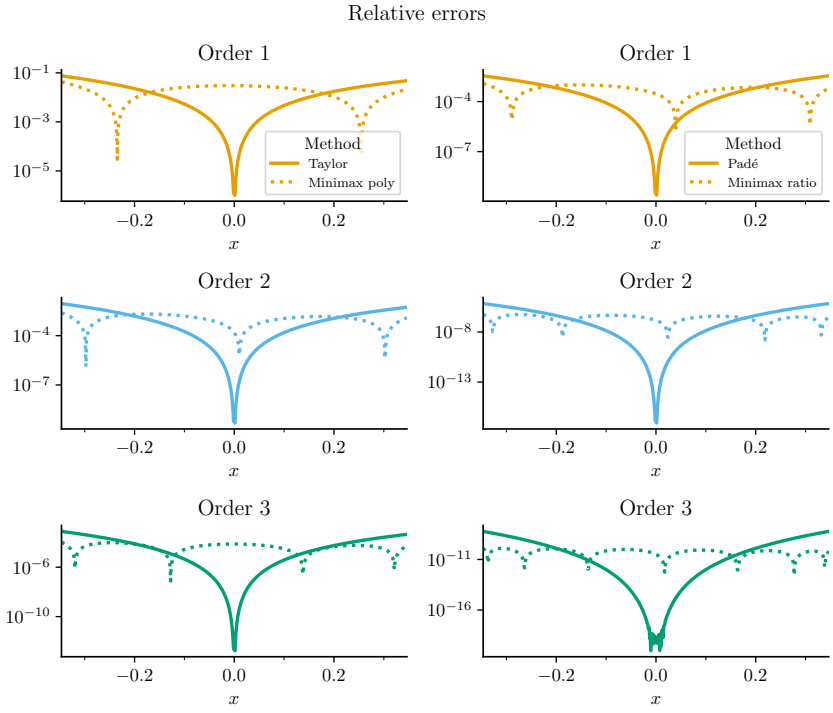


Figure 6: Relative errors of order-N Taylor- and minimax polynomials (left) and order-$[N/N]$ Padé- and minimax rational approximations (right).

[10] Unlike the Taylor and Padé approximators, the minimax implementations in this section utilize `mpmath` floating-point numbers rather than fixed-point numbers. This has no implications for the relative error analysis.

The comparison to local approximators shows how, for the same order, the minimax approaches "distribute" the errors across the interval, with each additional power term leading to a new peak in the relative errors. Meanwhile, the local approximators use every additional power term to increase the accuracy around zero. Minimax approximations, as expected, achieve lower maximal error, but higher minimal error; unlike the change from Taylor polynomials to Padé approximations, there are no shifts in the general order of magnitude of errors, but rather a redistribution of these within the chosen interval.

Since one is normally concerned primarily with the *worst case* performance of numerical approximation, minimax approaches are a popular choice utilized in the default implementations of many languages to get Padé-like orders of magnitude in errors, but distributed more evenly.[11]

For a given order N, the case for rational minimax approaches appears strong at first glance; however, several downsides weaken the case for fixed-point numbers in particular:

1. Unlike Padé coefficients, rational minimax coefficients do **not** have a "natural" integer representation. Thus additional work is required to identify an appropriate number of significant digits (which may be quite high) in order to rescale all coefficients to integers without causing overflow - or potentially utilize fixed-point coefficients of varying digits for each term, requiring an additional division.
2. Even with $N = M$, the symmetry of numerator- and denominator coefficients are lost in the minimax approach - and with it, the computational cost savings from the separation into even- and odd term sums in (3.11).
3. The multiplicative identity (3.12) for the highest-order coefficient can only be ensured in general by treating all other coefficients as fixed-point numbers (or likely accepting **significant** precision loss from truncation to integers).

Analogous to the points made in Section 3.3 comparing the computational cost of order-[2/3] and order-[3/3] Padé approximations, this motivates a comparison of the number of operations to determine if, once again, a Padé approximation of higher order can outperform the minimax approximation at a similar or lower number of operations.

The order-[2/2] and order-[3/3] minimax rational approximations[12]

$$\begin{aligned}
\hat{R}_{[2/2]}(x) &\approx \frac{11.95798930995 + 5.99100802161x + 1.00000000000x^2}{11.95798754936 - 5.96690830996x + 0.98806902258x^2} \\
&= \frac{1195798930995 + 599100802161x + 100000000000x^2}{1195798754936 - 596690830996x + 98806902258x^2}, \\
\hat{R}_{[3/3]}(x) &\approx \frac{119.665663565 + 59.884226173x + 11.989280835x^2 + 1.000000000x^3}{119.665663561 - 59.781437174x + 11.937886908x^2 - 0.991459909x^3} \\
&= \frac{119665663565 + 59884226173x + 11989280835x^2 + 1000000000x^3}{119665663561 - 59781437174x + 11937886908x^2 - 991459909x^3}
\end{aligned} \tag{3.22}$$

[11] This is especially relevant for functions with many local extrema, which may be poorly captured by local approximators – this is however **not** the case for the exponential function.

[12] The minimax coefficients are remarkably close to the Padé coefficients, hence the difference in behavior is due to *very small* differences in coefficients - meaning that the temptation to significantly truncate or round-off coefficients to the point of symmetry quickly deteriorates the approximations to a "worse" Padé approximation (unless rounded all the way to the original Padé coefficients).

are compared to Padé approximations from (3.14) in terms of operations required for evaluation of their integer-coefficient representations in Table 2.

	$+$	$-$	$\cdot$	$/$
$R_{[2/2]}$	2	1	3	2
$\hat{R}_{[2/2]}$	3	1	5	2
$R_{[3/3]}$	3	1	5	3
$\hat{R}_{[3/3]}$	4	2	9	3
$R_{[4/4]}$	4	1	7	4

Table 2: Number of operations in fixed-point Padé approximations $R_{[N/N]}$ and rational minimax approximations $\hat{R}_{[N/N]}$ assuming reuse of powers of x after rescaling.

Even in the best case of fully-integer representations of the minimax coefficients, the $R_{[3/3]}$ Padé approximation can be computed with just a single additional division compared to the $\hat{R}_{[2/2]}$ minimax approximation, while achieving several orders of magnitude lower maximal relative errors.

Similar to the case in the comparison between $[N/N]$- and $[N/M]$ Padé approximations, the $[N/N]$ Padé approximations once again scales better with increasing orders due to the sum separation (3.11) - going from the $\hat{R}_{[3/3]}$ minimax approximation to the $R_{[4/4]}$ Padé approximation *saves* one subtraction and two multiplications, still with only one additional division.

As can been seen from the comparison of maximal relative errors in Figure 7, if the goal is to stay below a certain "worst case" error, it is counterintuitively **not** optimal to use a minimax rational approximation in most cases, since using a higher-order Padé approximation will still yield lower maximal errors with similar or even fewer operations.

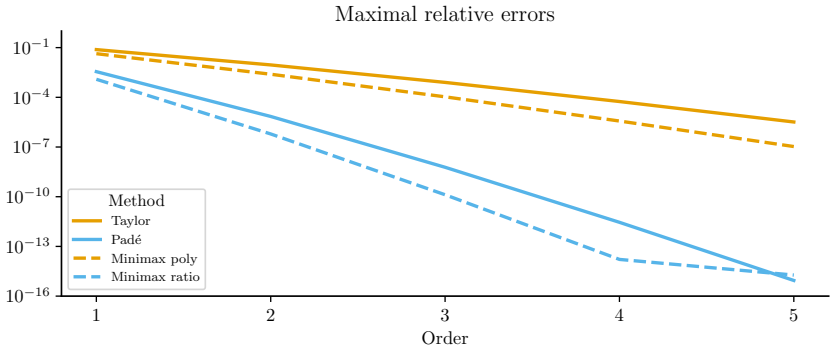


Figure 7: Maximal relative errors for $|x| \leq {}^1\!/_2 \log 2$ across order-N Taylor- and minimax polynomials and order-$[N/N]$ Padé- and minimax rational approximations.

The points made above in favor of Padé approximation for fixed-point number implementations are even slightly understated. In particular, we assumed a fully-integer representation was feasible for the minimax approximation, but computed the relative errors from an arbitrary-precision floating-point implementation - which would **understate** the errors if **any** truncation of coefficients took place.

We also ignored the additional complexity involved in obtaining the coefficients for the minimax approximations. Coefficients used here were computed via the `chebfun` package in `MATLAB` which, to the authors' knowledge, offers some of the most thoroughly-tested and state-of-the-art procedures for numerically stable floating-point approximations available - yet even more specialized implementations tailored to arbitrary-precision floating-point numbers would be required to find coefficients of higher order, since the order-[5/5] minimax routine already terminates early due to limitations in the machine precision of 64-bit floating-point numbers, as seen by the kink in Figure 7.[13]

Having covered the primary candidate methods for exponential function approximation, the following section provides a practical example for how one may configure a fixed-point approximation based on imposing requirements derived from a specific use case.

4 Approximator configuration: a practical example

With coverage of fixed-point numbers in Section 2 and various approximation methods for the exponential function in Section 3, we now give an example of arriving at a set configuration for the approximator order and fixed-point number representation based on practical requirements relevant to the application of the approximation.

Based on the review of approximation methods in Section 3, we utilize a bit-shifted order-$[N/N]$ Padé approximation. We want to compute the value of a deposited USD amount $\$C \geq 0$ accounting for compounded interest after $T \geq 0$ years for some fixed annual rate r_A. Since the lowest unit of USD is \$0.01 (one cent), this amount can be represented as a base-10 fixed-point number with $d_C = 2$ digits.

Note first that the annual rate r_A corresponds to a continous per-second rate of

$$r = \frac{\log\left(1 + r_A\right)}{\gamma} \tag{4.1}$$

with $\gamma = 60 \cdot 60 \cdot 24 \cdot 365$ being the number of seconds in a normal (non-leap) year such that after γ seconds, the deposit is worth

$$Ce^{r\gamma} = Ce^{\frac{\log(1+r_A)}{\gamma}\gamma} = C(1 + r_A). \tag{4.2}$$

We assume a dollar-amount error tolerance of at-most $\$\alpha$ for elapsed years $T \leq \beta$ implying for all $0 \leq x \leq r\gamma\beta = \beta \log(1 + r_A)$:

$$C\left|R_{[N/N]}(x) - e^x\right| \leq \alpha \Leftrightarrow \left|\frac{R_{[N/N]}(x) - e^x}{e^x}\right| \leq \frac{\alpha}{Ce^x} \leq \frac{\alpha}{C(1 + r_A)e^\beta}. \tag{4.3}$$

We thus accomplish the target error tolerance for $0 \leq x \leq r\gamma\beta$ with relative errors below the (constant) error threshold in (4.3), which depends on the the deposit amount C, the error tolerance amount α, the (annual) rate r_A and number of years β.

[13] See `https://github.com/chebfun/chebfun/blob/db207bc/minimax.m#L539`.

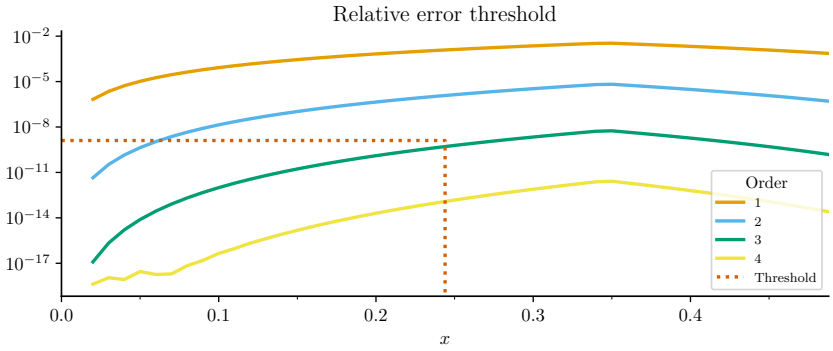


Figure 8: Relative errors of order-$[N/N]$ Padé approximations and threshold (4.3).

Figure 8 shows the threshold from (4.3) for parameters $\$C \coloneqq \$100\,000$, $\$\alpha \coloneqq \$0.02$, $r_A \coloneqq 0.05$ and $\beta \coloneqq 5$ across order-$[N/N]$ Padé approximation relative errors. We see that the order-[3/3] approximator achieves the target dollar amount error tolerance, since the relative errors stay below the threshold for $0 \leq x \leq r\gamma\beta \approx 0.24$.

Having identified an appropriate order for the approximator, we now turn to the tuning of the fixed-point number representation. Assume we want to support deposits up to $\bar{C} \gg C$: we choose some large value (ideally a safe amount above realistic inputs), e.g. $\bar{C} \coloneqq \$1\,000\,000\,000\,000 = \$10^{12}$ (one trillion). This amount as an (unsigned) fixed-point number in cents requires an additional two digits, so this deposit takes

$$\left\lceil \log_2 \left(\bar{C} \cdot 10^2 \right) \right\rceil = \left\lceil \log_2 10^{12+2} \right\rceil = \lceil 14 \log_2 10 \rceil = 47 \tag{4.4}$$

bits to represent.[14]

The approximator $R_{[3/3]}$ should support up to β years of compounding, in which case

$$R_{[3/3]}(x) \approx e^x \leq e^{r\gamma\beta} = e^{\log(1+r_A)\beta} = (1 + r_A)e^{\beta}. \tag{4.5}$$

The number of bits required to represent the approximations $R_{[3/3]}(x)$ as (unsigned) fixed-point numbers with $d_R \geq 0$ approximation digits is then at most

$$\left\lceil \log_2 \left(10^{d_R}(1 + r_A)e^{\beta} \right) \right\rceil = \left\lceil d_R \log_2 10 + \log_2(1 + r_A) + \beta \log_2 e \right\rceil. \tag{4.6}$$

Computing the deposit value after compounded rates requires the multiplication of the deposit value and the approximation, which then takes the number of bits corresponding to the sum of (4.4) and (4.6) to represent. Figure 9 shows the required bits for this product in the upper subplot, and the required bits for intermediary values relied on in the Padé approximator across approximation digits d_R in the lower subplot.

From Figure 9, we can see that choosing e.g. $d_R \coloneqq 16$ allows us to fit both the product (of the deposit $\bar{C}$ and approximation $R_{[3/3]}(r\gamma\beta)$) and the intermediary values relied on in the approximator within 128-bit integers, since the required bits will all be less than

[14] Implementing approximators that are safe for an arbitrary amount of a token on a blockchain would need to assume a value of the total (integer) supply of the minimal unit of the token in question.

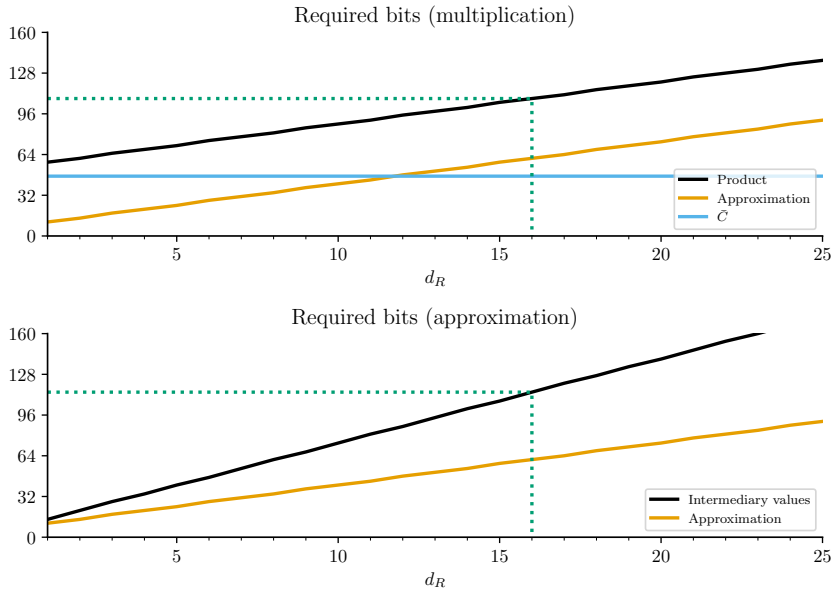


Figure 9: Required bits for the multiplication of a $\bar{C} = \$10^{12}$ deposit in cents with the approximation (upper) and the required bits for the approximator intermediary values (lower) across approximation digits d_R. Dotted line highlights $d_R := 16$.

128.[15] While the product will be rescaled to a smaller number after the multiplication (as discussed in Section 2), we cannot skip this initial computation of the numerator in general.

It is important to note that finding the bits required for the intermediary values requires assessing the maximal number of bits required across all points in the approximation process and across the *range* of values $0 \leq x \leq r\gamma\beta$; the bits in the intermediary values are **not** necessarily maximized at the largest input value.[16] One can attempt to derive these requirements more formally by finding bounds in each step of the approximation process, though here we took a more empirical approach via an integer subclass that keeps track of the maximal intermediary value after each operation.[17]

In practice the calibration process is often highly non-linear: it is typical to revisit the initial parameters and error tolerances after reviewing the required number of bits, tweaking and testing parameter changes until one arrives at a balance that "feels reasonable". The key point is to establish a systematic approach where one can directly observe the impact of configuration choices and guide any changes to this based on practical, application-specific considerations – ideally supported by automated tests to confirm the expected behavior.

Starting from "outrageously strict" requirements is often a **good** idea in the above approach: for example, if we needed to fit all values into a 64-bit environment, we may loosen the requirements on the supported deposit range, the per-second time resolution

[15]The gaps up to 128 bits then leaves additional space for e.g. larger deposits or longer compounding – but importantly we have a *guarantee* that the values implied from the chosen parameters can be represented without overflow with 128-bit integers.

[16]This may be amplified by the use of a bit-shifted approximator, though this does not take into effect in the ranges considered here as the range does not exceed $(\log 2)/2$.

[17]For further details, see the `github` repository linked in Section 3.1.

and dollar-amount errors to find that the order-[2/2] approximations could still provide adequate accuracy for our purposes. The benefit of starting from **overly** strict targets is that this leads to a "natural" continuous process of adjusting those parameters and error tolerances that may have been excessive at first until one finds an acceptable pragmatic trade-off that makes full use of the resources at hand (by only *barely* fitting into the available space).

The configuration arrived at in this section is by no means to be taken as a general recommendation, but rather to provide an example of how one can iterate towards a context-based calibration framework that derives a configuration from requirements that can be assessed in relation to the application.

5 Conclusion

This article provided a brief introduction to fixed-numbers and their context in Section 2 followed by coverage of a variety of methods for approximating the exponential function in Section 3 with notes specific to fixed-point numbers.

We showed how Padé approximations outperform Taylor polynomials when accounting for computational cost and introduced range reduction methods that utilize bit-shifting for achieving high accuracy across a wide range of inputs. Minimax methods were then covered, but ultimately found to not be competitive with Padé approximations for fixed-point numbers when accounting for savings in computational costs in symmetric order-$[N/N]$ Padé approximations.

Finally, Section 4 demonstrated an example of steps involved in constructing a calibration framework for fixed-point approximations that maps context-relevant parameters to the general configuration of the approximator.

References

Buterin, V. (2013). Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform.

Cody, W. J. and Ralston, A. (1967). A note on computing approximations to the exponential function. *Communications of the ACM*, 10(1):53–55.

Ehle, B. L. (1969). *On Padé approximations to the exponential function and A-stable methods for the numerical solution of initial value problems*. PhD thesis, University of Waterloo.

Filip, S.-I., Nakatsukasa, Y., Trefethen, L. N., and Beckermann, B. (2018). Rational minimax approximation via adaptive barycentric representations.

Horner, W. G. (1819). XXI. A new method of solving numerical equations of all orders, by continuous approximation. *Philos. Trans. R. Soc. Lond.*, 109(0):308–335.

Ismail, O. (1996). On multipoint Pade approximation for discrete interval systems. In *Proceedings of 28th Southeastern Symposium on System Theory*, SSST-96. IEEE Comput. Soc. Press.

Knuth, D. E. (1997). *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., USA.

Kudryavtsev, L. (2013). Taylor formula. *Encyclopedia of Math*. `https://web.archive.org/web/20230606030132/https://encyclopediaofmath.org/index.php?title=Taylor_formula&oldid=31209`. Archived: June 6, 2023.

Maehly, H. (1960). Approximations for the CDC 1604. *Control Data Corp.*

Moshier, S. L. (1992). Cephes Mathematical Library. `https://web.archive.org/web/20240508074714/https://www.netlib.org/cephes/`. Archived: May 8, 2024.

Pachon, R. and Trefethen, L. (2009). Barycentric-Remez algorithms for best polynomial approximation in the chebfun system. *BIT*, 49:721–741.

Padé, H. (1892). Sur la représentation approchée d'une fonction par des fractions rationnelles. *Annales scientifiques de l'École normale supérieure*, 9:3–93.

Pan, V. Y. (1966). Methods of computing values of polynomials. *Russian Mathematical Surveys*, 21(1):105–136.

Powell, M. J. D. (1981). *Approximation Theory and Methods*. Cambridge University Press.

Remez, E. Y. (1934). Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov*, 10(196):41–63.

Tang, P.-T. P. (1990). Accurate and efficient testing of the exponential and logarithm functions. *ACM Trans. Math. Softw.*, 16(3):185–200.

Taylor, B. (1717). *Methodus incrementorum directa & inversa*. Impensis Gulielmi Innys.

The mpmath development team (2023). *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0)*. `http://mpmath.org/`.

The University of Oxford and the Chebfun Developers (2024). Chebfun — numerical computing with functions. https://web.archive.org/web/20240515062214/https://www.chebfun.org. Archived: May 15, 2024.

Weisstein, E. W. (2023). Padé Approximant. From *MathWorld* – A Wolfram Web Resource. https://web.archive.org/web/20231203013322/https://mathworld.wolfram.com/PadeApproximant.html. Archived: December 3, 2023.

Wynn, P. (1966). On the Convergence and Stability of the Epsilon Algorithm. *SIAM Journal on Numerical Analysis*, 3(1):91–122.

Yates, R. (2020). Fixed-Point Arithmetic: An Introduction. *Digital Signal Labs*.

Automatic differentiation for diffusion operator integral variance reduction

Johan Auster*

Abstract

This paper demonstrates applications of automatic differentiation with nested dual numbers in the diffusion operator integral variance reduction framework originally proposed by Heath and Platen. Combining this estimator with automatic differentiation techniques for computing value function sensitivities allows for a flexible implementation without trade-offs in numerical stability or accuracy. This fully mitigates a key practical shortcoming of the original estimator, as we remove the dependency on error-prone and problem-specific manual calculations. We perform a relative error analysis of the estimator and standard Monte Carlo estimation against the numerical integration solution of the European call option in the Heston model and find computational time savings in excess of three orders of magnitude for the same expected relative errors for an at-the-money option. The implementation is further extended to the valuation of discrete down-and-out barrier call options and floating-strike lookback put options, demonstrating the relative ease of applying the automatic differentiation approach to path-dependent options with monitoring bias corrections.

Keywords: Monte Carlo, automatic differentiation, variance reduction, Heston model, barrier options, lookback options, monitoring bias.

JEL classification: C63, C65, G12.

*Department of Mathematical Sciences, University of Copenhagen, Denmark. E-mail: johan.auster@math.ku.dk.

1 Introduction

This paper demonstrates applications of automatic differentiation (AD) with nested dual numbers for computation of value function sensitivities in the diffusion operator integral (DOI) variance reduction framework introduced by Heath and Platen (2002) in their seminal paper.[1] Combining this estimator with AD methods allows for a significant reduction in the implementation complexity when applying the method across models and contracts, without trade-offs in numerical stability or accuracy of estimates.

The DOI estimator utilizes martingale properties of value functions, leading to a control variate-like sample-based method using a known value function evaluated at the initial time and a corrective integral running until a first exit time of the process. This integral depends on the sensitivities from the approximating value function applied to sample paths of the process from the original model, thereby incorporating information from the full path realization even when pricing non-path-dependent options. As shown in the original paper (Heath and Platen, 2002), this approach leads to a consistent and unbiased price estimator with drastic reductions in the variance of price estimates compared to simple averaging of realized discounted payoffs as in standard Monte Carlo (MC) methods.

In a follow-up paper by Heath and Platen (2014), DOI estimation was utilized in conjunction with partial differential equation (PDE) methods for a multidimensional diffusion model. In this approach, a smooth dimension-reduced approximation of the PDE solution to the pricing problem of interest is constructed via truncated Taylor series expansions. The approximating PDE is then combined with the DOI estimator to approximate the true solutions of European-style options in a three-component model representing a diversified equity index.

Recent works have illustrated further applications of this estimator to a variety of pricing problems and the significant variance reduction achievable over standard MC methods, e.g. in the valuation of short-rate derivatives in the Fong-Vasicek model (Coskun et al., 2019) and barrier options in the Heston model (Coskun and Korn, 2018), with the results of the latter being replicated in this paper.

A key practical limitation of the DOI approach is the reliance on second-order sensitivities for the approximating pricing problems, which quickly grow in complexity when dealing with non-constant variance approximations, volatility-dependent corrections for discrete path-dependent options and complicated value functions. By combining the dual number approach to differentiation with a general transformation of Black-Scholes value functions, sensitivities can be computed efficiently from single value function passes without the truncation and round-off errors of traditional numerical differentiation techniques. This enables applications of the estimator to a wide range of problems, requiring only the corresponding analytical value function from the Black-Scholes case.

This paper is organized as follows: in Section 2 we summarize the derivation of the DOI estimator and the form of the estimator in the Heston model with a suitable Black-Scholes approximation. Section 3 outlines the AD approach to exact differentiation with dual numbers, along with the extension to nested dual numbers for higher order differentiation. Section 4 presents numerical results and benchmarks of the estimator in the Heston

[1] Automatic differentiation is sometimes referred to as adjoint algorithmic differentiation, adjoint automatic differentiation, algorithmic differentiation, computational differentiation etc.

model, including a relative error analysis for the European call followed by applications to discrete down-and-out call barrier options and floating-strike lookback put options. Section 5 concludes.

2 The Heath-Platen DOI estimator

This section provides a summary of the general structure and derivation of the DOI estimator. The presentation is largely based on the original paper by Heath and Platen (2002), while leaving out some of the more intricate points. No new material or results are presented here and we refer to the original work for a more detailed derivation and discussion of the theoretical foundations of the estimator.

2.1 Setup

We let $T > 0$ and denote by Γ a path-connected subset of $\mathbb{R}^d$, and we let $X = (X_t)_{0 \leq t \leq T}$ be a real-valued d-dimensional Itô process starting from time $t = 0$ with initial value $X_0 = x \in \Gamma$ on the time horizon $[0, T]$ satisfying the stochastic integral equation

$$X_t = x + \int_0^t \mu(s, X_s)\,\mathrm{d}s + \int_0^t \sigma(s, X_s)\,\mathrm{d}W_s. \tag{2.1}$$

Here W is an m-dimensional standard Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, P, \mathbb{F})$, with the filtration $\mathbb{F}$ satisfying the usual conditions. μ and σ are $d \times 1$ and $d \times m$ matrices respectively, and we use the notation $\mu^i, \sigma^{i,j}$ to refer to the entries of the i'th row and j'th column of these, and we assume sufficient conditions on μ and σ for (2.1) to be a unique strong Markovian solution, e.g. linear growth bounds and Lipschitz continuity (Kloeden and Platen, 1992).

We introduce a stopping time $\tau : \Omega \to [0, T]$ given by

$$\tau \coloneqq \inf\{t \geq 0 : (t, X_t) \notin [0, T) \times \Gamma\}, \tag{2.2}$$

which is the first exit time of (t, X_t) from the region $[0, T) \times \Gamma$, and we define a payoff function $h : B \to \mathbb{R}$ with

$$B \coloneqq ([0, T) \times \partial\Gamma) \cup \left(\{T\} \times \overline{\Gamma}\right), \tag{2.3}$$

where $\partial\Gamma$ denotes the boundary of Γ and $\overline{\Gamma} = \Gamma \cup \partial\Gamma$ denotes the closure. The payoff is thus defined for paths in the exit region and at maturity, allowing for payoffs that may depend on a stochastic first exit time τ of the region Γ. This could correspond to the optimal exercise region for an American-style contract or a barrier knockout.

For pricing purposes we consider value functions $u(t, x) : [0, T] \times \Gamma \to \mathbb{R}$ of the form

$$u(t, x) \coloneqq \mathbb{E}_Q\left[h(\tau, X_\tau) \mid \mathcal{F}_t\right], \tag{2.4}$$

where we assume $u(t, x) \in \mathcal{C}^{1,2}([0, T] \times \Gamma)$, i.e. with $\partial_t u(t, x)$ continuous in t on $(0, T)$ for fixed $x \in \Gamma$ and $\partial_{x^i x^j} u(t, x)$ continuous in $x \in \Gamma$ for fixed $t \in (0, T)$ for all $i, j = 1, \ldots, d$. We use the notation ∂_x for the partial differentiation operator, e.g. $\partial_{tx} u(s, X_s)$ is the partial derivative of $u(t, x)$ w.r.t t and x evaluated at $t = s$ and $x = X_s$. Q is assumed

to be an appropriate risk-neutral measure in order for the market to be free of arbitrage opportunities in the no free lunch with vanishing risk (NFLVR) sense (Bjork, 2004).

Note that the value function in (2.4) can be interpreted either as a price function under some appropriately discounted dynamics or an undiscounted price function. Throughout the paper we use the latter interpretation.

From Itô's formula for continuous semimartingales (Øksendal, 2003), it holds that

$$\begin{aligned} u(\tau, X_\tau) = u(0,x) + \int_0^\tau \partial_t u(s, X_s)\,\mathrm{d}s + \int_0^\tau \sum_{i=1}^d \partial_{x^i} u(s, X_s)\,\mathrm{d}X_s^i \\ + \frac{1}{2}\int_0^\tau \sum_{i,j=1}^d \partial_{x^i x^j} u(s, X_s)\,\mathrm{d}\left[X^i, X^j\right]_s \end{aligned} \tag{2.5}$$

with $[X^i, X^j]$ denoting the quadratic covariation of X^i and X^j. From the definition of X in (2.1) we thus have

$$\begin{aligned} u(\tau, X_\tau) = u(0,x) + \int_0^\tau \partial_t u(s, X_s)\,\mathrm{d}s + \int_0^\tau \sum_{i=1}^d \mu^i(s, X_s)\,\partial_{x^i} u(s, X_s)\,\mathrm{d}s \\ + \int_0^\tau \sum_{i=1}^d \sum_{k=1}^m \sigma^{i,k}(s, X_s)\,\partial_{x^i} u(s, X_s)\,\mathrm{d}W_s^k \\ + \frac{1}{2}\int_0^\tau \sum_{i,j=1}^d \sum_{k=1}^m \sigma^{i,k}(s, X_s)\,\sigma^{j,k}(s, X_s)\,\partial_{x^i x^j} u(s, X_s)\,\mathrm{d}s, \end{aligned} \tag{2.6}$$

and interchanging summation and collecting terms then gives

$$u(\tau, X_\tau) = u(0,x) + \int_0^\tau \mathcal{L}^0 u(s, X_s)\,\mathrm{d}s + \sum_{k=1}^m \int_0^\tau \mathcal{L}^k u(s, X_s)\,\mathrm{d}W_s^k \tag{2.7}$$

using the operator notation

$$\begin{aligned} \mathcal{L}^0 \coloneqq \partial_t + \sum_{i=1}^d \mu^i(t,x)\,\partial_{x^i} + \frac{1}{2}\sum_{i,j=1}^d \sum_{k=1}^m \sigma^{i,k}(t,x)\,\sigma^{j,k}(t,x)\,\partial_{x^i x^j}, \\ \mathcal{L}^k \coloneqq \sum_{i=1}^d \sigma^{i,k}(t,x)\,\partial_{x^i}. \end{aligned} \tag{2.8}$$

The operator $\mathcal{L}^0$ is the "diffusion operator", which is the namesake of the estimator presented in the following sections.

2.2 Uncoupled DOI estimator

Assume further that the payoff $h : B \to \mathbb{R}_+$ has sufficient regularity such that the process

$$\mathbb{E}_Q\left[h(\tau, X_\tau) \mid \mathcal{F}_t\right] = u(t \wedge \tau, X_{t\wedge\tau}) \tag{2.9}$$

is a square-integrable martingale; since τ is a bounded stopping time, this depends only on the regularity of h. We then consider some approximating value function $\bar{u}(t,x)$ with the boundary condition

$$\bar{u}(\tau, X_\tau) = u(\tau, X_\tau) = h(\tau, X_\tau) \tag{2.10}$$

with $\bar{u}(t,x) \in \mathcal{C}^{1,2}([0,T]\times\Gamma)$. Furthermore assume that $\int_0^{t\wedge\tau} \mathcal{L}^k \bar{u}\,(s, X_s)\,\mathrm{d}W_s^k$ is a square-integrable martingale null at zero for all $k = 1,\dots,m$, hence in particular zero in expectation. Then from the boundary condition (2.10) and applying the representation (2.7) to $\bar{u}(t,x)$ we have that

$$\begin{aligned} u\,(0,x) &= \mathbb{E}_Q\left[h\,(\tau, X_\tau) \mid X_0 = x\right] \\ &= \mathbb{E}_Q\left[\bar{u}\,(\tau, X_\tau) \mid X_0 = x\right] \\ &= \bar{u}\,(0,x) + \mathbb{E}_Q\left[\int_0^\tau \mathcal{L}^0 \bar{u}\,(s, X_s)\,\mathrm{d}s \mid X_0 = x\right], \end{aligned} \tag{2.11}$$

from which the "uncoupled" DOI estimator results from (discounted) realizations of (2.11). The following section shows how this estimator can be extended to incorporate approximating model dynamics.

2.3 DOI estimator with approximating dynamics

Consider now another d-dimensional stochastic process $\bar{X}$ approximating the dynamics of X in some sense and starting from the same initial values $x \in \Gamma$ with the same regularity assumptions as for (2.1). From Itô's formula we once again have

$$\bar{u}\left(\tau, \bar{X}_\tau\right) = \bar{u}\,(0,x) + \int_0^\tau \bar{\mathcal{L}}^0 \bar{u}\left(s, \bar{X}_s\right)\mathrm{d}s + \sum_{k=1}^m \int_0^\tau \bar{\mathcal{L}}^k \bar{u}\left(s, \bar{X}_s\right)\mathrm{d}W_s^k \tag{2.12}$$

with $\bar{\mathcal{L}}$ operators analogous to the ones defined in (2.8). Taking expectations in (2.12) and using the martingale property of $\bar{u}(t,x)$ leads to the Kolmogorov backward PDE

$$\bar{\mathcal{L}}^0 \bar{u}\,(t,x) = 0 \tag{2.13}$$

on $[0,T]\times\Gamma$ with $\bar{u}(t,x) = h(t,x)$ on B, hence

$$\mathbb{E}_Q\left[\int_0^\tau \bar{\mathcal{L}}^0 \bar{u}\,(s, X_s)\,\mathrm{d}s \mid X_0 = x\right] = 0. \tag{2.14}$$

Subtracting (2.14) from the final equation in (2.11) then leads to the coupled DOI estimator:

$$Z_\tau \coloneqq \bar{u}\,(0,x) + \int_0^\tau \left(\mathcal{L}^0 - \bar{\mathcal{L}}^0\right)\bar{u}\,(s, X_s)\,\mathrm{d}s. \tag{2.15}$$

The $\mathcal{L}^0 - \bar{\mathcal{L}}^0$ integral couples the two processes via their diffusion operators, and from (2.15) we can immediately observe the mechanics behind the resulting variance reduction: provided we have the solution to $\bar{u}\,(0,x)$, we can think of this as a prior from our approximating model, while the variance of the correcting integral term is low when the operators $\mathcal{L}^0$ and $\bar{\mathcal{L}}^0$ cancel, which is exactly the case when $\bar{X}$ closely mimicks the dynamics of X. Interestingly, the integral term only depends on the true value function implicitly through the $\mathcal{L}^0$ diffusion operator, as the operators are applied only to the approximating $\bar{u}$ function. The practical application of the estimator is discussed further in Section 2.4 and Section 4.1.

The above estimator can be furthed generalized, as the boundary condition (2.10) is not technically necessary. In this case, one would replace the estimator in (2.15) with

$$\tilde{Z}_\tau \coloneqq \bar{u}\,(0,x) + h\,(\tau, X_\tau) - \bar{u}\,(\tau, X_\tau) + \int_0^\tau \left(\mathcal{L}^0 - \bar{\mathcal{L}}^0\right)\bar{u}\,(s, X_s)\,\mathrm{d}s, \tag{2.16}$$

where the additional $h(\tau, X_\tau) - \bar{u}(\tau, X_\tau)$ term corrects for any difference in payoffs. This allows for a more flexible choice of approximating contracts, for example one may choose a European put approximation when pricing the American-style counterpart.

2.4 Application in the Heston model

We now apply the DOI estimator in (2.15) to the Heston model. The Heston model (Heston, 1993) consists of the two stochastic differential equations (SDEs):

$$\begin{aligned} \mathrm{d}S_t &= S_t \left(r \,\mathrm{d}t + \sqrt{\nu_t} \,\mathrm{d}W_t \right) \\ \mathrm{d}\nu_t &= \kappa \left(\theta - \nu_t \right) \mathrm{d}t + \xi \sqrt{\nu_t} \,\mathrm{d}Z_t \end{aligned} \tag{2.17}$$

starting from initial values $S_0, \nu_0 > 0$ with a constant risk-free rate $r \in \mathbb{R}$ and strictly positive constants κ, θ and ξ. W and Z are Brownian motions with correlation $\rho \in [-1, 1]$ such that $Z \overset{\mathrm{d}}{=} \rho W + \sqrt{1-\rho^2}\tilde{W}$ for Brownian motions $W, \tilde{W}$ with $\tilde{W} \perp W$, and the Feller condition $2\kappa\theta > \xi^2$ is assumed to ensure that the variance process ν a.s. remains positive.

As an approximating process we use the same Generalized Black-Scholes (GBS) model as proposed by Heath and Platen (2002) with dynamics

$$\begin{aligned} \mathrm{d}\bar{S}_t &= \bar{S}_t \left(r \,\mathrm{d}t + \sqrt{\bar{\nu}_t} \,\mathrm{d}W_t \right) \\ \mathrm{d}\bar{\nu}_t &= \kappa \left(\theta - \bar{\nu}_t \right) \mathrm{d}t \end{aligned} \tag{2.18}$$

with W a Brownian motion and parameters matching those in (2.17). The (2.18) dynamics are resemblant of the Heston model, but with only the deterministic mean reversion ordinary differential equation (ODE) component in the variance process.

From the definition in (2.8) we have

$$\begin{aligned} \mathcal{L}^0 &= rS\partial_S + \frac{1}{2}\nu S^2 \partial_{SS} + \kappa \left(\theta - \nu \right) \partial_\nu + \xi\nu \left(\rho S \partial_{S\nu} + \frac{1}{2}\xi \partial_{\nu\nu} \right), \\ \bar{\mathcal{L}}^0 &= rS\partial_S + \frac{1}{2}\nu S^2 \partial_{SS} + \kappa \left(\theta - \nu \right) \partial_\nu \end{aligned} \tag{2.19}$$

such that the ∂_S and ∂_{SS} terms cancel in (2.15) and we are left with

$$\mathcal{L}^0 - \bar{\mathcal{L}}^0 = \xi\nu \left(\rho S \partial_{S\nu} + \frac{1}{2}\xi \partial_{\nu\nu} \right). \tag{2.20}$$

Value functions in the approximating GBS model can then be fully characterized by Black-Scholes value functions through a simple volatility transformation presented in Section 6.1. Note that the resulting diffusion operator term (2.20) was already presented in the original paper (Heath and Platen, 2002).

In practice the differenced diffusion operator in (2.20) is evaluated at discrete time points for path realizations of (2.17). The DOI estimator hence incorporates information from the approximating GBS dynamics though the known value function along each discrete point. In contrast, many control variate techniques only incorporate known quantities statically.[2] Section 4 illustrates in greater detail the advantageous properties that result from this utilization of path data for valuation, regardless of the terminal payoff for a particular sample.

[2] A standard example is to add and subtract S_T from the standard MC estimator when pricing a call option and rewriting the positive S_T term as the known expectation at the initial time (Glasserman, 2003).

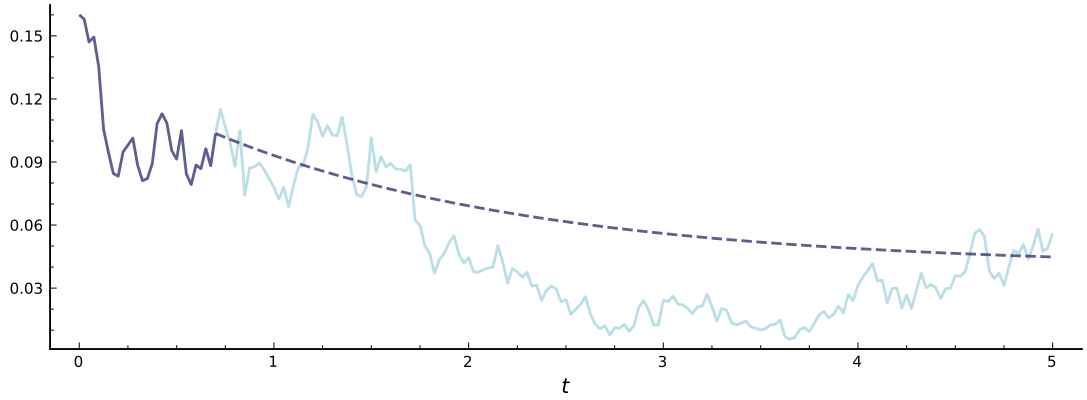


Figure 1: Realization of ν (solid line) and approximating ODE $\bar{\nu}$ (dashed line).

3 Exact differentiation with nested dual numbers

In this section we outline the core idea behind differentiation with dual numbers and their extension to nested dual numbers for higher-order differentiation, and we demonstrate how these are used to retrieve the relevant value function sensitivities in (2.20).

Dual number AD overcomes several of the shortcomings of finite difference (FD) methods, as derivatives can be computed exactly in a single function pass with limited computational overhead, while simultaneously avoiding the numerical instability from standard FD differentiation.[3]

AD is increasingly being adopted in favor of more traditional FD methods (Baydin et al., 2018) and are used in the implementations of gradient descent algorithms in many popular machine learning libraries, including `PyTorch`, `TensorFlow` and `Flux.jl`. AD methods have also seen recent attention within applications specific to finance, e.g. for computing greeks for discontinuous payoff functions from MC estimates (Daluiso and Facchinetti, 2018). For a detailed discussion of AD methods, see Griewank and Walther (2008).

3.1 Dual numbers

A dual number $z \coloneqq a + b\varepsilon$ is characterized by a real 2-tuple $(a, b) \in \mathbb{R}^2$ with the property that

$$\varepsilon^2 = 0, \tag{3.1}$$

which forms a unital associative commutative algebra over the reals, allowing us to extend real functions to functions of dual numbers in a general way.[4]

[3] To quote Baydin et al. (2018), FD numerical differentiation schemes commit a cardinal sin of numerical analysis: "thou shalt not subtract numbers which are approximately equal".

[4] In so many words, we have an algebraic structure with well-defined commutative and associative addition and multiplication, and for any $z = a + b\varepsilon$ we have a zero element $0 + 0\varepsilon$, additive inverse $-a - b\varepsilon$, multiplicative identity $1 + 0\varepsilon$ and distributivity s.t $(z_1 + z_2) \cdot (z_3 + z_4) = z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4$.

For any function f real analytic at $a \in \mathbb{R}$ with n'th derivative $f^{(n)}$, the Taylor series of f around a evaluated at $a + b\varepsilon$ is given by

$$\begin{aligned} f(a + b\varepsilon) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (a + b\varepsilon - a)^n \\ &= f(a) + \sum_{n=1}^{\infty} \frac{f^{(n)}(a)}{n!} b^n \varepsilon^n \\ &= f(a) + f^{(1)}(a)\, b\varepsilon \end{aligned} \tag{3.2}$$

from the property in (3.1), since $\varepsilon^n = 0$ for all $n \geq 2$. Setting $b = 1$ and evaluating coefficients of the real part $f(a)$ and non-real part $f^{(1)}(a)$ separately then yields the function value at a and first-order derivative at a respectively. This is not an approximation of the derivative, but the exact derivative; and this with only a single evaluation of the function.[5]

For two dual numbers $z_1 \coloneqq a_1 + b_1\varepsilon$ and $z_2 \coloneqq a_2 + b_2\varepsilon$, we have the operations

$$\begin{aligned} z_1 \pm z_2 &= (a_1 \pm a_2) + (b_1 \pm b_2)\, \varepsilon, \\ z_1 \cdot z_2 &= a_1 a_2 + (a_1 b_2 + a_2 b_1)\, \varepsilon, \\ \frac{z_1}{z_2} &= \frac{a_1}{a_2} + \frac{b_1 a_2 - a_1 b_2}{a_2^2} \varepsilon. \end{aligned} \tag{3.3}$$

Here the ε terms can be seen to align with basic rules of differentiation, and naturally this further extends to, for example, the chain rule for composite functions. In practice, dual numbers are propagated through series of simple operations when function values are computed, from which the above properties lead to general, efficient and exact differentiation without the need for manually specifying analytical derivative solutions to any terms or relying on potentially numerically unstable and computationally expensive FD schemes.

3.2 Higher-order differentiation with nested dual numbers

In order to apply dual numbers to higher-order differentiation problems, we further extend our notion of dual numbers to the case of nested dual numbers, where an N-dimensional nested dual number is given by

$$z \coloneqq a + \sum_{n=1}^{N} b_n \varepsilon_n \tag{3.4}$$

with $a \in \mathbb{R}$ and where any of the b_n coefficients may themselves be nested dual numbers. In analogy with (3.1), we impose that

$$\varepsilon_n^2 = \varepsilon_m^2 = (\varepsilon_n \varepsilon_m)^2 = 0 \tag{3.5}$$

for all $n, m = 1, \ldots, N$, noting that this property does not imply $\varepsilon_n = 0$, $\varepsilon_m = 0$ nor $\varepsilon_n \varepsilon_m = 0$, as these are not real numbers.

[5] This is suggestive of the potential for gains in computational efficiency compared to manual differentiation in cases where analytical derivatives are computationally expensive to evaluate.

The definition in (3.4) includes as a special case the following two-dimensional nested dual number:

$$\begin{aligned} z^h &\coloneqq a + b_1\varepsilon_1 + (b_2 + b_3\varepsilon_1)\,\varepsilon_2 \\ &= a + b_1\varepsilon_1 + b_2\varepsilon_2 + b_3\varepsilon_1\varepsilon_2 \end{aligned} \tag{3.6}$$

such that the Taylor series of f real analytic at $a \in \mathbb{R}$ evaluated at z^h is

$$\begin{aligned} f\left(z^h\right) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}\left(z^h - a\right)^n \\ &= f(a) + f^{(1)}(a)\left(b_1\varepsilon_1 + b_2\varepsilon_2 + b_3\varepsilon_1\varepsilon_2\right) + \frac{1}{2}f^{(2)}(a)\left(2b_1b_2\varepsilon_1\varepsilon_2\right) \\ &= f(a) + f^{(1)}(a)\,b_1\varepsilon_1 + f^{(1)}(a)\,b_2\varepsilon_2 + f^{(1)}(a)\,b_3\varepsilon_1\varepsilon_2 + f^{(2)}(a)\,b_1b_2\varepsilon_1\varepsilon_2 \end{aligned} \tag{3.7}$$

from expanding terms and applying (3.5), after which we retrieve the first and second order derivatives by evaluating the appropriate coefficients for the non-real parts as in (3.2) with $b_1 = b_2 = 1$ and $b_3 = 0$. The collection of dual numbers of the form z^h in (3.6) is sometimes referred to as the hyper-dual numbers; these are discussed in depth in Fike and Alonso (2011).

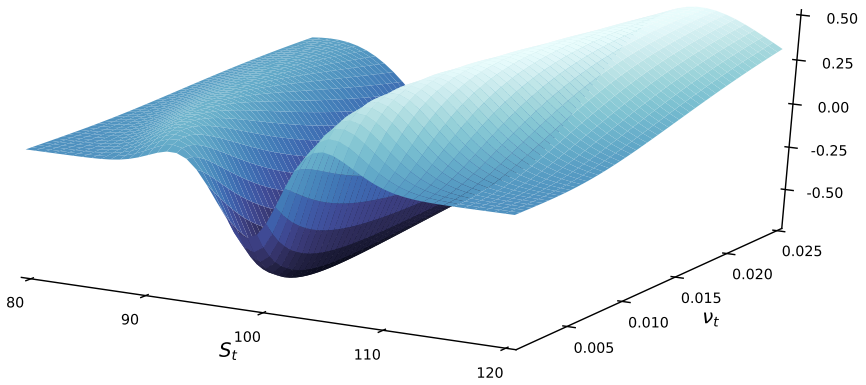
Finally, to see how the nested dual numbers enable us to retrieve the required sensitivites for our application in (2.20), we consider a value function $u(t,x)$ with $t \in [0,T]$ for $T > 0$ and $x \coloneqq (x^1, x^2) \in \mathbb{R}^2$ and define the nested dual numbers

$$\begin{aligned} z^1 &\coloneqq \left(x^1 + 1\varepsilon_1 + 0\varepsilon_2\right) + (0 + 0\varepsilon_1 + 0\varepsilon_2)\,\varepsilon_3, \\ z^2 &\coloneqq \left(x^2 + 0\varepsilon_1 + 1\varepsilon_2\right) + (1 + 0\varepsilon_1 + 0\varepsilon_2)\,\varepsilon_3. \end{aligned} \tag{3.8}$$

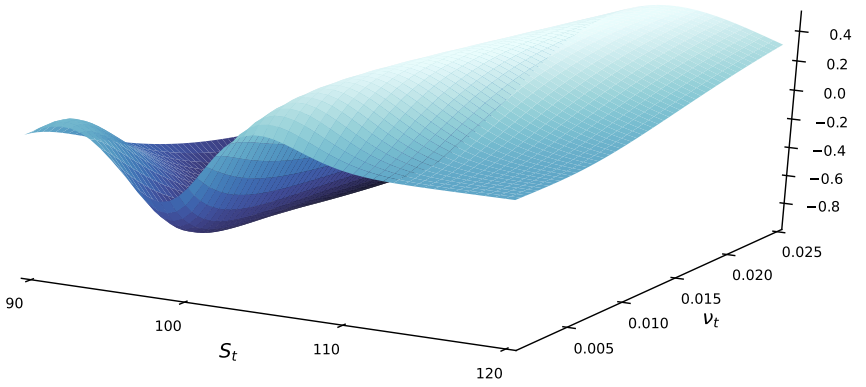
Then the Taylor series of $u(t,x)$ around x evaluated at $z \coloneqq (z^1, z^2)$ gives

$$\begin{aligned} u(t,z) = &\left(u(t,x) + \partial_{x^1}u(t,x)\,\varepsilon_1 + \partial_{x^2}u(t,x)\,\varepsilon_2\right) \\ &+ \left(\partial_{x^2}u(t,x) + \partial_{x^1x^2}u(t,x)\,\varepsilon_1 + \partial_{x^2x^2}u(t,x)\,\varepsilon_2\right)\varepsilon_3, \end{aligned} \tag{3.9}$$

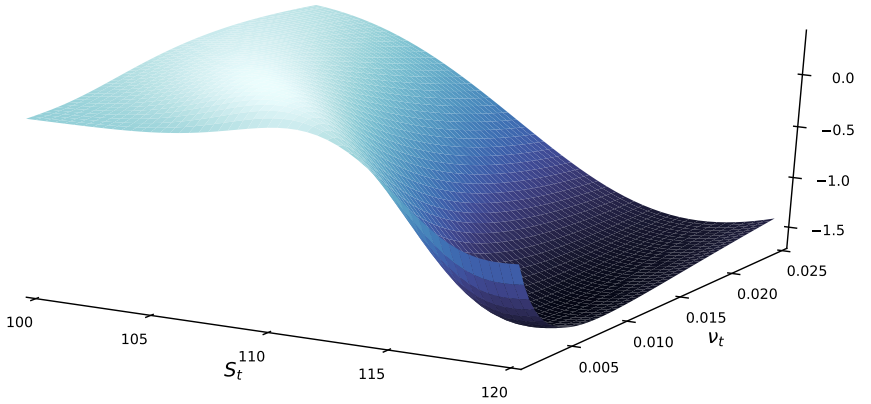
from which the sensitivities $\partial_{s\nu}u$ and $\partial_{\nu\nu}u$ needed in (2.20) are exactly the coefficients associated with the two non-real terms $\varepsilon_1\varepsilon_3$ and $\varepsilon_2\varepsilon_3$ from the nesting dual number.



(a) Call / put with $K = 100$



(b) Down-and-out barrier call with $K = 100, H = 90$ and 200 monitoring points



(c) Floating-strike lookback put with $\max_{0 \le u < t} S_u = 120$

Figure 2: Differenced diffusion operators $\mathcal{L}^0 - \bar{\mathcal{L}}^0$ for the Heston model computed with dual numbers at $t = 0.7$ for the European call / put, discretely-monitored down-and-out barrier call and continuous floating-strike lookback put with parameter set **A**. Puts and calls have the same Vanna and Vomma in the European case (Haug, 2007), so their diffusion operators are identical.

4 Numerical results

This section presents the details of the implementation and the results of various benchmarks of the estimator in the Heston model. First we compare the AD DOI estimates of the European call option to the standard non-AD method with manually derived sensitivities and confirm that the resulting estimates are equivalent. Estimate errors are then assessed against the numerical integration solution and the relationship between errors and moneyness is investigated.

The estimator is then applied to the pricing problem of the discrete down-and-out barrier call option by utilizing monitoring bias correction (Broadie et al., 1997) of the value function for the continuous option, with results compared against the mean estimates and confidence intervals achieved with the non-AD method in the paper of Coskun and Korn (2018).

Finally we apply the estimator to the valuation of the continuously-monitored floating-strike lookback put option, demonstrating the apparent lack of any practically significant monitoring bias in contrast to the standard MC estimator. We then extend the application to the discrete counterpart of this option type through a correction of the value function (Broadie et al., 1999) similar to the discrete barrier option.

4.1 Implementation details

The estimator is implemented in `Julia 1.4.1` (Bezanson et al., 2017) and the full source code is available online on `https://github.com/austerj/ad-doi`. All benchmarks are run on a laptop with an `Intel Core i7-8650U 1.90GHz` CPU running `Ubuntu 20.04 LTS`. The `Dual` number type from the `ForwardDiff.jl` package (Revels et al., 2016) is used for automatic differentiation of the value function sensitivities. All random numbers are generated using the `xoroshiro128+` algorithm (Blackman and Vigna, 2018) as implemented in the `RandomNumbers.jl` package.

For generation of sample paths, a Predictor-Corrector (PC) scheme with degrees of implictness 0.5 for the drift and no diffusion implicitness[6] (Kloeden and Platen, 1992) is implemented with correctors

$$\begin{aligned}\hat{S}_{n+1} &\coloneqq \hat{S}_n + \frac{\Delta}{2} r \left(\tilde{S}_{n+1} + \hat{S}_n\right) + \hat{S}_n \sqrt{\hat{\nu}_n} \Delta W_n \\ \hat{\nu}_{n+1} &\coloneqq \hat{\nu}_n + \Delta\kappa \left(\theta - \frac{1}{2}\left(\tilde{\nu}_{n+1} + \hat{\nu}_n\right)\right) + \xi \sqrt{\hat{\nu}_n} \Delta Z_n\end{aligned} \tag{4.1}$$

for Euler-Maruyama predictors

$$\begin{aligned}\tilde{S}_{n+1} &\coloneqq \hat{S}_n \left(1 + \Delta r + \sqrt{\hat{\nu}_n} \Delta W\right) \\ \tilde{\nu}_{n+1} &\coloneqq \hat{\nu}_n + \Delta\kappa \left(\theta - \hat{\nu}_n\right) + \xi \sqrt{\hat{\nu}_n} \Delta Z\end{aligned} \tag{4.2}$$

with $\Delta \coloneqq T/N$ denoting the step size for maturity $T > 0$ and number of steps $N \in \mathbb{N}$ and $\Delta W, \Delta Z$ normal random variables with variance Δ and correlation $\rho \in [-1, 1]$. Variance process samples $\hat{\nu}$ are truncated at 10^{-8} to prevent negative values. The scheme

[6] While the inclusion of diffusion implicitness can lead to a highly efficient scheme for small N, numerical instability can arise as the number of time steps increases (Platen and Shi, 2008).

is initialized at the initial values of the Heston underlying asset and variance process, i.e. $(\hat{S}_0, \hat{\nu}_0) \coloneqq (S_0, \nu_0)$, and the differenced diffusion operator in (2.20) is integrated numerically using the trapezoidal rule.

The baseline model parameters for the Heston model are chosen in accordance with parameter set **A** defined in Table 1, which are themselves based on the parameters in (Heath and Platen, 2002) with a larger initial variance ν_0. These parameters are used throughout the numerical section unless deviations are explicitly stated, i.e. in tables with varying parameters and in Section 4.4. All curves fitted to the resulting estimates and their errors are attained using standard ordinary least squares (OLS) methods with the noted transformations of dependent variables.

S_0	ν_0	r	κ	θ	ξ	ρ	T
100	0.16	0.04	0.6	0.04	0.2	-0.15	1

Table 1: Parameter set **A** used as a baseline throughout the numerical section.

Listing 1 shows the code for the implementation of the single-threaded DOI estimator for non-path-dependent contracts. A multi-threaded version of the estimator has been implemented in addition, with each thread using a distinct random number generator with a random unsigned 128-bit integer as a seed. The latter estimator is used in the proceeding benchmarks due to the significant performance benefits over the single-threaded version.

While subtle complications, such as false sharing, can arise from random number generation on multiple threads (Hellekalek, 1998), the multi-threaded implementation passes the two-sample Kolmogorov-Smirnov test at a 99.9% confidence level against the single-threaded estimator, i.e. estimates produced by the two algorithms are identically distributed. The test is available on the `github` repository within the `test` folder.

Listing 1: Single-threaded implementation of the AD DOI estimator

```
function estimator(nsteps, npaths, model, contract, rng)
    @unpack s₀, ν₀, r, κ, θ, ξ, ρ = model
    @unpack T = contract
    Δ = T/nsteps
    t = Δ:Δ:T
    u₀ = u(0., s₀, ν₀, model, contract)  # computes σ̄ and transforms BS to GBS
    A₀ = diffop(0., s₀, ν₀, model, contract)/2  # first point of trapezoidal
    payoffs = Vector{Float64}(undef, npaths)
    doi = Vector{Float64}(undef, npaths)
    for j = 1:npaths
        s, ν = s₀, ν₀
        A = A₀
        for i = 1:nsteps
            s, ν = step(s, ν, Δ, model, rng)
            A += diffop(t[i], s, ν, model, contract)
        end
        # diffop is zero at time T, hence division not needed for trapezoidal
        A *= Δ
        payoffs[j] = h(s, contract)
        doi[j] = A
    end
    discount = exp(-r*T)
    mc_estimate = discount * mean(payoffs)
    doi_estimate = discount * (u₀ + mean(doi))
    mc_estimate, doi_estimate
end
```

```
function diffop(t, s, ν, model, contract)
    @unpack r, κ, θ, ξ, ρ = model
    ∂sν, ∂νν = sensitivities(t, s, ν, model, contract, state)
    ξ*ν*(ρ*s*∂sν + 0.5*ξ*∂νν)
end

function sensitivities(t, s, ν, model, contract, state)
    zs = Dual(Dual(s, 1., 0.), Dual(0., 0., 0.))
    zν = Dual(Dual(ν, 0., 1.), Dual(1., 0., 0.))
    uz = u(t, zs, zν, model, contract, state)
    ∂sν = uz.partials[1].partials[1]  # vanna
    ∂νν = uz.partials[1].partials[2]  # vomma
    ∂sν, ∂νν
end
```

4.2 Analytical and AD comparison for European call

We first consider the valuation problem of a European call option in the Heston model. This contract provides a payoff of

$$h\left(S_T, T\right) \coloneqq \max\left(S_T - K, 0\right) = \left(S_T - K\right)^+ \tag{4.3}$$

at the time of maturity $T > 0$ with strike price $K > 0$.

This contract has the well-known closed-form time t value function solution in the Black-Scholes framework:

$$u^{\mathrm{BS}}\left(t, x\right) = xe^{r(T-t)}\Phi\left(d_1\right) - K\Phi\left(d_2\right) \tag{4.4}$$

where Φ denotes the standard normal cumulative distribution function, and with

$$\begin{aligned} d_1 &\coloneqq \frac{\log\frac{x}{K} + \left(r + \frac{1}{2}\sigma^2\right)\left(T - t\right)}{\sigma\sqrt{T-t}}, \\ d_2 &\coloneqq d_1 - \sigma\sqrt{T-t} \end{aligned} \tag{4.5}$$

starting from initial value of the underlying $x > 0$ with constant volatility $\sigma > 0$ and risk-free rate $r \in \mathbb{R}$. This is then transformed into the appropriate GBS value function as shown Section 6.1, from which the value sensitivities relied on in (2.20) are computed via dual numbers in the AD method and analytically in the standard method. The derivation of the analytical diffusion operator expression is shown in Section 6.2.

In the comparison of the two implementations of the DOI estimator, 200 time steps are used for each sample path of the Heston dynamics based on parameter set **A** and we use a strike price of $K = 100$.

Figure 3 shows 100 realizations of the European call payoff (4.3) against single-path estimates for each of the DOI estimator implementations, as well as both DOI estimators against each other. As expected, the analytical DOI and AD (dual) DOI estimates are perfectly aligned for every path, showing that the value sensitivity computation via dual numbers indeed produces the same result as direct evaluation of analytical solutions.

Furthermore, Figure 3 showcases an imperfect correlation between payoffs and DOI estimates, exemplifying concretely the ability of the estimator to utilize path realization data beyond the terminal payoff. Additionally, the range of DOI estimates is noteworthy; even single-path estimates are contained in a tight band, with a distance of only 0.7 between the minimum and maximum compared to approximately 120 for payoff realizations.

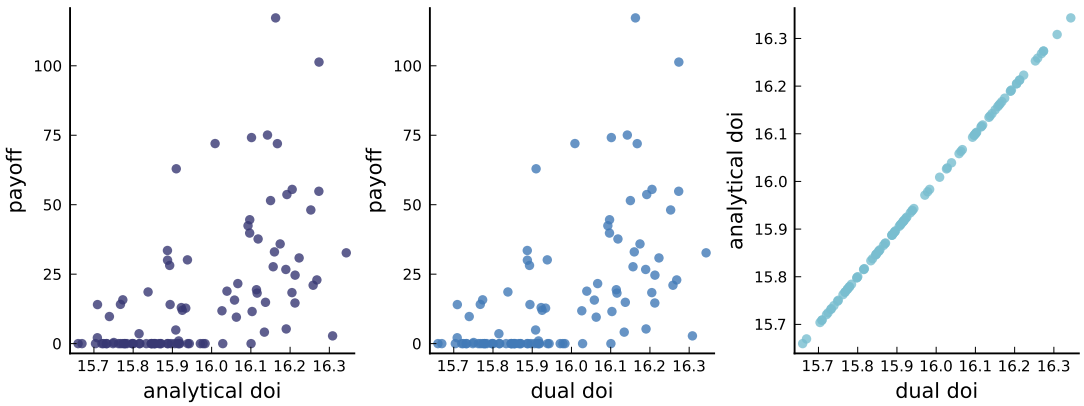


Figure 3: Scatter plots of European call payoff realizations against both DOI estimates and AD DOI estimates against analytical DOI estimates for the same underlying paths.

For the remainder of the numerical section, we exclusively use the AD DOI method and simply refer to this as the DOI estimator.

4.3 Relative error analysis and pricing of European call

Relative errors (REs) for the DOI and MC estimates are measured via the semi-closed-form solution computed with numerical integration (Heston, 1993), first for a single strike in order to assess the performance of the DOI estimator across numbers of sample paths, then across a range of strike prices to assess how REs scale with moneyness.

Using 200 time steps for the simulated paths, we estimate prices for the European call with $K = 100$ and with all other parameters as specified in parameter set **A** in Table 1. We use $\lfloor 10^n \rfloor$ sample paths for $n = 1, 1.25, \ldots, 5.75, 6$, where $\lfloor x \rfloor$ denotes the floor operator. The resulting mean REs are shown in Figure 4 from running the DOI and MC estimators 50 times for each number of sample paths.

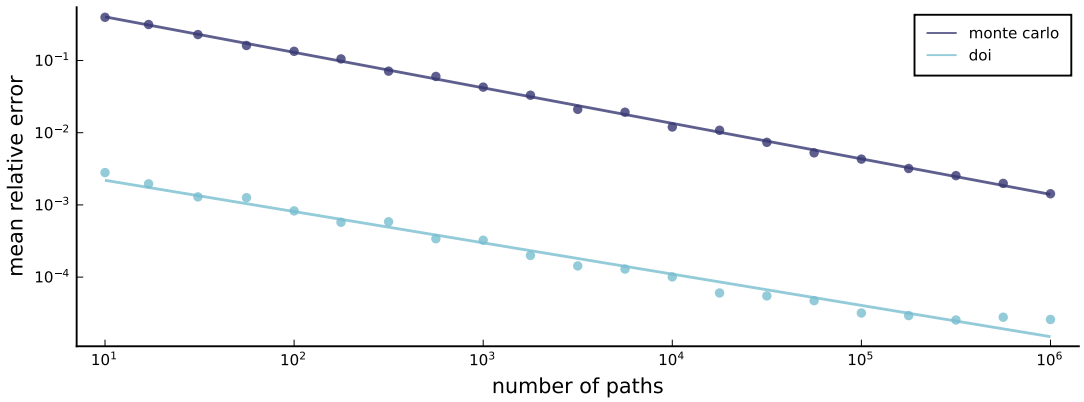


Figure 4: Mean relative errors for European call price estimates for the DOI and MC estimators in a $\log_{10}$-$\log_{10}$ plot as a function of number of paths M. Both MC and DOI estimates are fitted to $\log_{10} M$.

While the convergence rate of the DOI estimator suggested by the OLS fit is comparable to the standard MC convergence, the variance reduction is apparent in the shift in REs: the mean RE of 0.28% for the DOI estimator with 10 sample paths corresponds to

the mean RE from approximately 250 000 paths with the regular MC estimator. A benchmark run on the same laptop used throughout this paper revealed that for multi-threaded implementations of both estimators at these configurations, the MC estimator had a median runtime of 384.0 milliseconds compared to just 161.5 microseconds for the DOI estimator, i.e. computational time gains of a factor of almost 2 400 for comparable expected REs with 200 time steps.[7]

We stress that the large variance reduction does not solely come from the initial GBS price, as directly taking the GBS price of 16.74 as an estimate would lead to a RE of 5.1% against the Heston price of 15.94 in this case. A significant dynamic correction is thus incorporated from each sample via the integral term in (2.15), demonstrating that the DOI estimator is able to efficiently incorporate new data for the target model price from even a small number of paths.

Next we analyze how the performance of the estimator scales over a range of strikes. Using the same parameters as before, we run both estimators 10 000 times with 200 time steps and 10 sample paths per run across each of the strike prices $K = 20, 25, \ldots, 175, 180$. The resulting mean REs are presented in Figure 5.

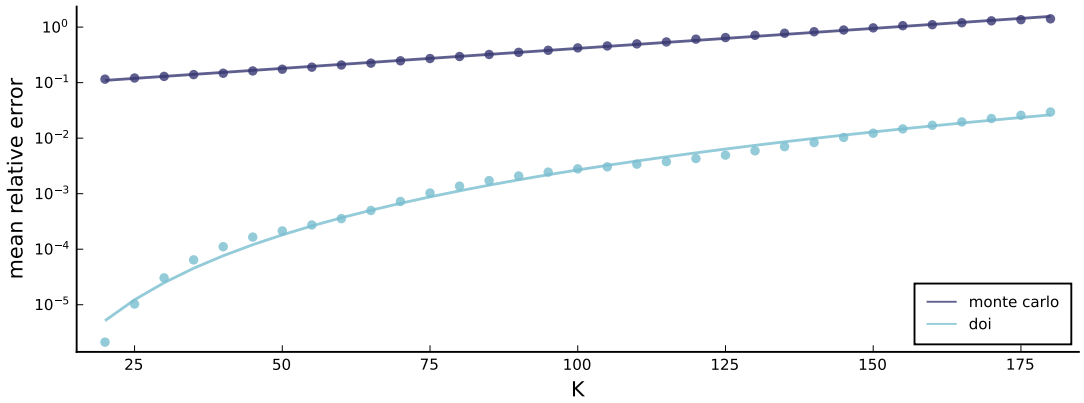


Figure 5: Mean relative errors for European call price estimates for the DOI and MC estimators in a semi-$\log_{10}$ plot across strike prices K. MC estimates are fitted to K, DOI estimates are fitted to $\log_{10} K$.

While the order of magnitude of REs of standard MC estimates grow linearly in strike prices, the DOI REs grows sublinearly. Consistent with the RE analysis across sample paths, the REs for the MC estimator are larger by orders of magnitude, in particular for lower strike prices.

Table 2 outlines summary statistics of the MC and DOI estimator after 50 runs with 10 000 sample paths per estimate and 200 time steps. Parameter set **A** is used as a baseline, but using the noted values for initial variance ν_0. Consistent with the RE analysis, confidence intervals of the DOI estimates are significantly narrower than for the MC estimates across strikes and initial variances, while the mean estimates produced by the two estimators are closely aligned.

[7]Even more extreme factors could likely be achieved for in-the-money contracts as indicated by Figure 5 and by using fewer time steps for a diffusion-implicit PC scheme in the DOI estimation.

		MC		AD DOI	
K	ν_0	Mean	Conf.Interval	Mean	Conf.Interval
90	0.01	14.4970	(14.2735,14.7205)	14.4867	(14.4842,14.4893)
	0.04	16.1085	(15.7597,16.4574)	16.0981	(16.0947,16.1014)
	0.16	21.1127	(20.4906,21.7349)	21.1253	(21.1217,21.1289)
95	0.01	10.5287	(10.3314,10.7261)	10.5541	(10.5507,10.5575)
	0.04	12.6923	(12.3826,13.0021)	12.6921	(12.6883,12.6959)
	0.16	18.3987	(17.8175,18.9798)	18.3925	(18.3889,18.3961)
100	0.01	7.2057	(7.0166,7.3949)	7.1990	(7.1958,7.2022)
	0.04	9.7673	(9.4823,10.0523)	9.7563	(9.7522,9.7603)
	0.16	15.9465	(15.3980,16.4950)	15.9401	(15.9370,15.9433)
105	0.01	4.5641	(4.4133,4.7149)	4.5851	(4.5820,4.5883)
	0.04	7.3414	(7.1037,7.5791)	7.3198	(7.3164,7.3231)
	0.16	13.7448	(13.1701,14.3195)	13.7597	(13.7562,13.7631)
110	0.01	2.7614	(2.6379,2.8849)	2.7591	(2.7568,2.7613)
	0.04	5.3952	(5.1621,5.6282)	5.3747	(5.3713,5.3780)
	0.16	11.8874	(11.4560,12.3188)	11.8349	(11.8316,11.8382)

Table 2: Mean estimates and 95% confidence intervals for the European call option for the standard MC estimator and the DOI estimator. Parameter set **A** is used as a baseline.

4.4 Pricing of discrete down-and-out call barrier

Based on the paper by Coskun and Korn (2018), we now apply the estimator to the pricing problem of a discretely-monitored down-and-out call barrier option. Down-and-out call barrier contracts give a payoff of

$$h\left(S_T, T\right) \coloneqq \left(S_T - K\right)^+ \mathbf{1}_{\{m_T > H\}}, \quad m_T \coloneqq \min_{u \in \mathcal{T}} S_u \tag{4.6}$$

at maturity $T > 0$ with strike $K > 0$, barrier $H > 0$ and with $\mathcal{T} \subseteq [0, T]$ being the set of monitoring times. The payoff is thus that of a European call option under the additional condition that the underlying asset is not below the barrier H at any monitored time during the lifetime of the contract. The continuosly-monitored barrier value function, i.e. taking $\mathcal{T} = [0, T]$, has an analytical Black-Scholes solution (Haug, 2007)

$$u^{\mathrm{BS}}\left(t, x\right) = x e^{r(T-t)} \left(\Phi\left(d_1\right) - \Phi\left(h_1\right) \left(\frac{H}{x}\right)^{p+1} \right) - K \left(\Phi\left(d_2\right) - \Phi\left(h_2\right) \left(\frac{H}{x}\right)^{p-1} \right) \tag{4.7}$$

for $x \geq H$ with

$$
\begin{aligned}
d_1 &\coloneqq \begin{cases} \frac{\log \frac{x}{K} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, & H < K \\ \frac{\log \frac{x}{H} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, & H \geq K \end{cases}, \\
h_1 &\coloneqq \begin{cases} \frac{\log \frac{H^2}{xK} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, & H < K \\ \frac{\log \frac{H}{x} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, & H \geq K \end{cases}, \\
d_2 &\coloneqq d_1 - \sigma\sqrt{T-t}, \\
h_2 &\coloneqq h_1 - \sigma\sqrt{T-t}, \\
p &\coloneqq \frac{2r}{\sigma^2}.
\end{aligned}
\tag{4.8}
$$

When utilizing standard MC methods, we are implicitly dealing with discrete monitoring of the barrier, as one cannot observe the potential breach of the barrier between discrete points in time. However, as shown by Broadie et al. (1997), for a barrier H and N equidistant discrete monitoring points, there exists the following relationship between discrete barrier value functions $u_N^{\mathrm{BS}}(H)$ and continuous barrier value functions $u^{\mathrm{BS}}(H)$:

$$
u_N^{\mathrm{BS}}(H) = u^{\mathrm{BS}}\left(He^{\pm\beta\sigma\sqrt{T/N}}\right) + o\left(\frac{1}{\sqrt{N}}\right) \tag{4.9}
$$

using $+\beta$ for $H > x$ (e.g. up-and-out barriers) and $-\beta$ otherwise (e.g. our application for the down-and-out barrier) with the constant $\beta \coloneqq {-\zeta\left(\frac{1}{2}\right)}/{\sqrt{2\pi}} \approx 0.5826$ for the Riemann zeta function ζ. We utilize this corrected barrier level H at each pass of the value function for the DOI estimator to account for the discrete monitoring. Since (4.9) is a function of σ, the adjusted barrier must be recomputed at each evaluation of the value function. For the equivalent analytical implementation of the DOI estimator, the dependence on σ in (4.9) would have to be accounted for in the manually derived sensitivities.

In order to get comparative results to the paper of Coskun and Korn (2018), we utilize parameter set **B** outlined in Table 3. Additionally, we use a stepsize of $\Delta = 0.004$ leading to the number of time steps $N = {T}/{\Delta} = 125$. As expressed in (2.15), if the barrier H is crossed before maturity T, the DOI estimate from the given path will include the integration term up until the barrier crossing at time $\tau \leq T$.

S_0	ν_0	r	κ	θ	ξ	ρ	T
100	0.04	0.04	0.6	0.04	0.2	-0.8	0.5

Table 3: Parameter set **B** used for the discrete barrier option.

Table 4 shows summary statistics across strikes K and barriers H from 50 runs of the estimator for each configuration with 10 000 sample paths per estimate. The Coskun-Korn column contains the corresponding summary statistics as reported by Coskun and Korn (2018) for the same number of sample paths.

The aforementioned paper employs a truncated Euler-Maruyama scheme, which may explain the slight differences in estimates compared to the drift-implicit PC scheme in (4.1). Regardless, the AD DOI and MC mean estimates are consistent across parameters.

		MC		AD DOI		Coskun-Korn	
K	H	Mean	Conf.Interval	Mean	Conf.Interval	Mean	Conf.Interval
90	92	10.6707	(10.4032,10.9382)	10.6629	(10.6609,10.6650)	10.7489	(10.7466,10.7512)
	95	8.3302	(8.1118,8.5485)	8.3085	(8.3044,8.3126)	8.3899	(8.3847,8.3951)
	98	4.6752	(4.4552,4.8952)	4.6963	(4.6904,4.7022)	4.7526	(4.7454,4.7598)
100	92	5.7400	(5.5662,5.9139)	5.7427	(5.7369,5.7486)	5.6910	(5.6849,5.6971)
	95	4.7567	(4.5985,4.9148)	4.7178	(4.7142,4.7214)	4.6761	(4.6724,4.6799)
	98	2.8242	(2.7039,2.9445)	2.8266	(2.8255,2.8276)	2.8158	(2.8147,2.8168)
110	92	2.1084	(2.0214,2.1955)	2.1026	(2.0961,2.1091)	2.0608	(2.0538,2.0679)
	95	1.8225	(1.7409,1.9041)	1.8248	(1.8182,1.8315)	1.7750	(1.7678,1.7823)
	98	1.1757	(1.1026,1.2487)	1.1832	(1.1774,1.1891)	1.1523	(1.1459,1.1587)

Table 4: Mean estimates and 95% confidence intervals for the discrete up-and-out barrier call option with 125 monitoring points for the standard MC estimator, the AD DOI estimator and results from the standard DOI estimator implementation from Coskun and Korn (2018). Parameter set **B** is used for all estimates.

As with the European call, Table 4 shows a significant variance reduction over the MC estimator. The width of MC confidence bands are upwards of 130 times the width of the DOI confidence bands for the highly in-the-money case ($K = 90, H = 92$), and still more than 12 times for all out-of-the-money options ($K = 110$).

4.5 Monitoring bias and pricing of floating-strike lookback put

We finally apply the DOI estimator to the valuation of a lookback put option with a floating strike price. This contract gives a terminal payoff of

$$h\left(S_T, T\right) \coloneqq \max_{u \in \mathcal{T}} S_u - S_T \tag{4.10}$$

at maturity $T > 0$ with the set of monitoring points $\mathcal{T} \subseteq [0, T]$, and hence depends on the maximal value attained at monitoring points throughout the horizon of the contract. The floating strike refers to the fact that this payoff is identical to a European put option with a strike equal to this maximal attained value of the underlying across the monitored times of the underlying asset.

The arbitrage-free value function of the continuously-monitored contract with $\mathcal{T} = [0, T]$ in the Black-Scholes model is

$$u^{\mathrm{BS}}\left(t, x\right) = -xe^{r(T-t)}\Phi\left(-d_1\right) + m\Phi\left(-d_2\right) + \frac{x\sigma^2}{2r}\left(e^{r(T-t)}\Phi\left(d_1\right) - \left(\frac{m}{x}\right)^{\frac{2r}{\sigma^2}}\Phi\left(d_3\right)\right) \tag{4.11}$$

using the same notation as before and assuming $r > 0$, and with

$$\begin{aligned} m &\coloneqq \max_{0 \le u \le t} S_u, \\ d_1 &\coloneqq \frac{\log \frac{x}{m} + \left(r + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, \\ d_2 &\coloneqq d_1 - \sigma\sqrt{T-t}, \\ d_3 &\coloneqq d_1 - \frac{2r\sqrt{T-t}}{\sigma}. \end{aligned} \tag{4.12}$$

The pricing formula in (4.11) was originally proven by Goldman et al. (1979), and an alternative derivation can be found in chapter 6 of the book by Musiela (2005). As before, this value function is readily transformed into the GBS value function by the volatility transformation shown in Section 6.1. Besides the tracking of the rolling maximum m for each path, no further results or additions to the implementation are necessary to utilize the AD implementation for this contract.

When pricing contracts dependent on a continuously monitored extremal value of the underlying, a monitoring bias occurs when using standard MC methods without additional bias corrections. In the case of the lookback put, a negative bias from monitoring the rolling maximum on a discretized simulation would arise, as a higher payoff from attaining a new maximum between two points cannot be observed.

This bias improves relatively slowly in the number of steps $N \in \mathbb{N}$ for MC methods relying on a standard discretization of the underlying dynamics. While methods to overcome the monitoring bias problem for lookback options have been proposed, e.g. by Andersen and Brotherton-Ratcliffe (1996), such bias correction methods can be model specific and in some cases have lead to overcorrections producing even larger biases with opposite signs as discussed in Linetsky (2004).

We again use parameter set **A** for all model parameters and run both the DOI and MC estimators using 100 000 sample paths per estimate across number of steps $N = 2^n$ for $n = 4, 5, \ldots, 20, 21$ in order to assess the impact on price estimation from the number of steps used per sample path.

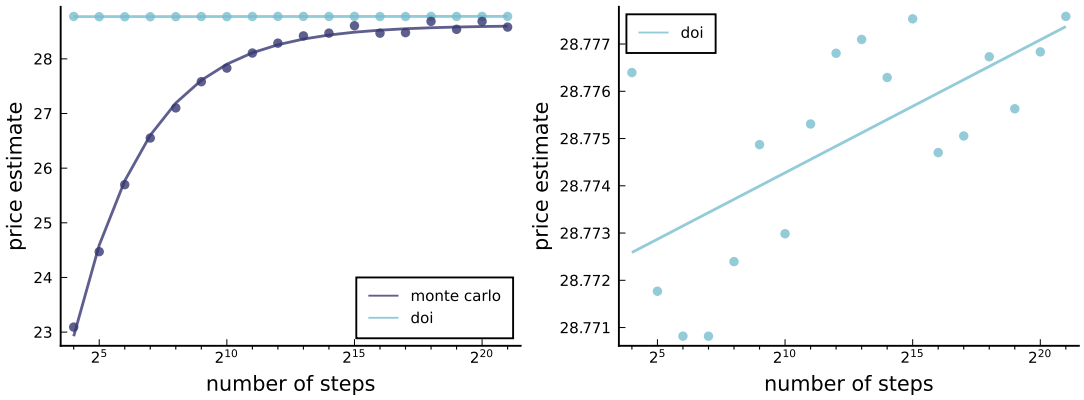


Figure 6: Price estimates for floating-strike lookback puts for the DOI and MC estimators in semi-$\log_2$ plots as functions of number of steps N. MC estimates are fitted to $1/\sqrt{N}$, DOI estimates are fitted to $\log_2 N$.

The resulting MC price estimates in Figure 6 clearly reveal the large monitoring bias, even for impractically large numbers of time steps. The DOI estimates however do not show a comparable bias, being practically constant regardless of the number of time steps, as the method can leverage the analytical value function for the continuous lookback option.

In particular, the increase from the smallest to the largest observed DOI estimate is below 0.03%, in stark contrast to an almost 24% increase from the MC estimate at 2^4 steps to the 2^{21}-step estimate. This close clustering of DOI estimates in absolute terms suggests that, unlike the MC estimator, any possible monitoring bias is of limited practical concern.

In order to get a comparable pricing problem for the MC and DOI estimators, we now take the same approach as in Section 4.4 by utilizing the following relationship shown by Broadie et al. (1999) between the discrete floating-strike lookback put value function u_N^{BS} with N equidistant monitoring points and the continuous value function u^{BS}:

$$u_N^{\mathrm{BS}}(m) = e^{-\beta\sigma\sqrt{T/N}} u^{\mathrm{BS}}\left(me^{\beta\sigma\sqrt{T/N}}\right) + xe^{r(T-t)}\left(e^{-\beta\sigma\sqrt{T/N}} - 1\right) + o\left(\frac{1}{\sqrt{N}}\right) \tag{4.13}$$

where β is defined as in Section 4.4 and x, m refers to the underlying value and (discretely) observed maximum respectively. By applying the above transformation to the value function, we obtain a variance reduced estimator of the discrete problem. As with the barrier option, replicating this approach with the non-AD method requires manual derivation of the sensitivities of the transformed value function (4.13).

		MC		AD DOI	
S_0	ν_0	Mean	Conf.Interval	Mean	Conf.Interval
90	0.04	11.7428	(11.5299,11.9557)	11.7021	(11.6977,11.7064)
	0.16	24.1983	(23.8682,24.5283)	24.2618	(24.2559,24.2677)
	0.36	38.4869	(38.0183,38.9556)	38.5792	(38.5739,38.5844)
95	0.04	12.4053	(12.2282,12.5824)	12.3527	(12.3476,12.3579)
	0.16	25.5533	(25.2243,25.8823)	25.6093	(25.6051,25.6135)
	0.36	40.6457	(40.1154,41.1760)	40.7234	(40.7165,40.7303)
100	0.04	13.0540	(12.8480,13.2600)	13.0023	(12.9965,13.0082)
	0.16	26.8630	(26.5003,27.2257)	26.9580	(26.9533,26.9628)
	0.36	42.8005	(42.2420,43.3589)	42.8666	(42.8596,42.8736)
105	0.04	13.6964	(13.4576,13.9352)	13.6522	(13.6468,13.6575)
	0.16	28.1811	(27.8040,28.5581)	28.3054	(28.2986,28.3123)
	0.36	44.9436	(44.4033,45.4838)	45.0092	(45.0026,45.0158)
110	0.04	14.3338	(14.1287,14.5389)	14.3023	(14.2967,14.3079)
	0.16	29.5346	(29.1753,29.8938)	29.6533	(29.6476,29.6590)
	0.36	47.0962	(46.5446,47.6477)	47.1536	(47.1470,47.1602)

Table 5: Mean estimates and 95% confidence intervals for the discrete floating-strike lookback put option with 200 monitoring points for the standard MC estimator and the DOI estimator. Parameter set **A** is used as a baseline.

Table 5 shows summary statistics across initial values of the underlying S_0 and intial variances ν_0 with 200 time steps, 10 000 sample paths per estimate and 50 runs for each set of parameters. After applying the bias correction transformation, the mean estimates of the MC and DOI estimates are now closely aligned across model parameters, while the reduction in confidence intervals is consistent with the previous pricing problems.

5 Conclusion

This paper demonstrated how the use of AD methods can augment the DOI estimator introduced by Heath and Platen by allowing for efficient and numerically stable evaluation of value sensitivities with dual numbers, while mitigating the complications involved in applying the estimator to general valuation problems, as only the Black-Scholes price and a suitable transformation are needed to achieve significant variance reduction.

We benchmarked the estimator in the valuation of European calls in the Heston model, where we saw an improvement of orders of magnitude in REs for the same number of paths compared with the standard MC estimator, with gains in computational time exceeding three orders of magnitude for the same expected REs in the benchmarked at-the-money option.

The implementation was further applied to the valuation of the discrete down-and-out barrier call, where the monitoring bias correction impact on sensitivities was effectively handled by the dual number approach. Finally, the estimator was applied to both continuous and discrete floating-strike lookback put options. Close clustering of price estimates in the continuous case across a large range of number of steps indicate that the monitoring bias, which severely limits effective valuation with traditional MC methods for this class of options, is of limited practical concern.

6 Appendix

6.1 Black-Scholes with transformed volatility

Value functions in the GBS model can be fully characterized by Black-Scholes value functions with transformed volatilities; to see this, note that for the GBS dynamics (2.18) starting from strictly positive initial values $(\bar{S}_t, \bar{\nu}_t) = (x^1, x^2)$ on the finite horizon $[t, T]$ with $T > t$, we have that

$$\bar{S}_{t,T} = x^1 e^{r(T-t) - \frac{1}{2}\int_t^T \bar{\nu}_{t,s}\,\mathrm{d}s + \int_t^T \sqrt{\bar{\nu}_{t,s}}\,\mathrm{d}W_s}, \tag{6.1}$$

for a Q-Brownian motion W.

By the Dambis-Dubins-Schwarz theorem (Revuz and Yor, 1999), we can represent the stochastic integral in (6.1) as a time-changed Brownian motion

$$\begin{aligned} \int_t^T \sqrt{\bar{\nu}_{t,s}}\,\mathrm{d}W_s &\overset{\mathrm{d}}{=} W_{\int_t^T \bar{\nu}_{t,s}\,\mathrm{d}s} \\ &\overset{\mathrm{d}}{=} \sqrt{\frac{\int_t^T \hat{\nu}_{t,s}\,\mathrm{d}s}{T-t}} W_{T-t}, \end{aligned} \tag{6.2}$$

hence the analytical solution to the approximating variance ODE[8] shows that

$$\bar{S}_{t,T} \overset{\mathrm{d}}{=} x^1 e^{\left(r - \frac{1}{2}\bar{\sigma}^2_{t,T}\right)(T-t) + \bar{\sigma}_{t,T} W_{T-t}} \tag{6.3}$$

[8]This is a first order linear inhomogeneous ODE with constant coefficients, hence the analytical solution is $\bar{\nu}_{t,T} = \theta + (\bar{\nu}_t - \theta)e^{-\kappa(T-t)}$ s.t. $\int_t^T \bar{\nu}_{t,s}\,\mathrm{d}s = \theta(T-t) + \frac{\bar{\nu}_t - \theta}{\kappa}\left(1 - e^{-\kappa(T-t)}\right)$.

with

$$\bar{\sigma}_{t,T}^2 \coloneqq \theta + \frac{x^2 - \theta}{\kappa(T-t)} \left(1 - e^{-\kappa(T-t)}\right). \tag{6.4}$$

The representation in (6.3) directly implies that a value function $\bar{u}(t,x)$ in the GBS model starting from values $x = (x^1, x^2)$ can be expressed equivalently as

$$\begin{aligned} \bar{u}(t,x) &= \mathbb{E}_Q \left[h(T, S_T) \mid \bar{S}_t = x_1, \bar{\nu}_t = x_2 \right] \\ &= u^{\mathrm{BS}}_{\bar{\sigma}_{t,T}}(t, x^1) \end{aligned} \tag{6.5}$$

for the corresponding Black-Scholes value function $u^{\mathrm{BS}}_{\bar{\sigma}}(t, x^1)$ with modified volatility $\bar{\sigma}_{t,T}$ computed as in (6.4) from the initial value $\bar{\nu}_t = x^2$ for the deterministic variance process.

While this example is based on the approximating dynamics for the Heston model (2.18), the above approach can be applied in exactly the same manner for a wide variety of approximating variance processes with analytical solutions to drift ODEs, e.g. multifactor Heston models (Gourieroux and Sufana, 2004), the 3/2 geometric mean reversion model proposed by Platen (1997) or the multifactor model of Grasselli (2016).

6.2 Analytical diffusion operator for the European call

The analytical solution to the (differenced) diffusion operators in (2.20) for the European call option can be derived from computing the derivatives of the value function (4.4) after applying the transformation outlined in Section 6.1.

Due to non-constant volatility in the GBS model, the chain rule is required in order to account for the time-varying volatility dynamics. In particular, we have

$$\begin{aligned} \partial_\nu \bar{u} &= \partial_{\bar{\sigma}} u^{BS}_{\bar{\sigma}} \cdot \partial_\nu \bar{\sigma}, \\ \partial_{\nu\nu} \bar{u} &= \partial_{\bar{\sigma}\bar{\sigma}} u^{BS}_{\bar{\sigma}} \cdot (\partial_\nu \bar{\sigma})^2 + \partial_{\bar{\sigma}} u^{BS}_{\bar{\sigma}} \cdot \partial_{\nu\nu} \bar{\sigma}, \\ \partial_{S\nu} \bar{u} &= \partial_{S\bar{\sigma}} u^{BS}_{\bar{\sigma}} \cdot \partial_\nu \bar{\sigma} \end{aligned} \tag{6.6}$$

with ν denoting the realization of the stochastic volatility process (2.17) used as the starting value of the approximating deterministic volatility ODE at each point in time. Since the approximating volatility satisfies

$$\bar{\sigma} = \bar{\sigma}_{t,T} = \sqrt{\theta + \frac{\nu - \theta}{\kappa\,(T-t)} \left(1 - e^{-\kappa(T-t)}\right)} \tag{6.7}$$

it follows that

$$\partial_\nu \bar{\sigma} = \frac{1 - e^{-\kappa(T-t)}}{2\kappa\,(T-t)} \frac{1}{\bar{\sigma}} \tag{6.8}$$

and subsequently

$$\begin{aligned} \partial_{\nu\nu} \bar{\sigma} &= \frac{1 - e^{-\kappa(T-t)}}{2\kappa\,(T-t)} \partial_\nu \left[\frac{1}{\bar{\sigma}}\right] \\ &= -\frac{1 - e^{-\kappa(T-t)}}{2\kappa\,(T-t)} \frac{1}{\sigma^2} \partial_\nu \bar{\sigma} \\ &= -\frac{(\partial_\nu \bar{\sigma})^2}{\bar{\sigma}}. \end{aligned} \tag{6.9}$$

Letting φ denote the standard normal density, we have the well-known sensitivies from the Black-Scholes model (Haug, 2007):

$$\begin{aligned}
\partial_{\bar{\sigma}} u_{\bar{\sigma}}^{BS} &= e^{r(T-t)} S\varphi\left(d_1\right)\sqrt{T-t}, \\
\partial_{\bar{\sigma}\bar{\sigma}} u_{\bar{\sigma}}^{BS} &= \partial_{\bar{\sigma}} u_{\bar{\sigma}}^{BS} \frac{d_1 d_2}{\bar{\sigma}}, \\
\partial_{S\bar{\sigma}} u_{\bar{\sigma}}^{BS} &= -e^{r(T-t)} \varphi\left(d_1\right) \frac{d_2}{\bar{\sigma}}.
\end{aligned} \tag{6.10}$$

Substituting these into (6.6) and simplifying, we finally have

$$\begin{aligned}
\partial_{\nu\nu} \bar{u} &= e^{r(T-t)} S\varphi\left(d_1\right) \frac{d_1 d_2 - 1}{\bar{\sigma}^3} \frac{\left(1 - e^{-\kappa(T-t)}\right)^2}{4\kappa^2 \left(T-t\right)^{3/2}}, \\
\partial_{S\nu} \bar{u} &= -e^{r(T-t)} \varphi\left(d_1\right) \frac{d_2}{\bar{\sigma}^2} \frac{1 - e^{-\kappa(T-t)}}{2\kappa\left(T-t\right)},
\end{aligned} \tag{6.11}$$

which gives us the necessary terms in (2.20).

References

Andersen, L. and Brotherton-Ratcliffe, R. (1996). Exact exotics. *Risk*, 9:85–89.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98.

Bjork, T. (2004). *Arbitrage Theory in Continuous Time*. Oxford University Press.

Blackman, D. and Vigna, S. (2018). Scrambled Linear Pseudorandom Number Generators. *CoRR*, abs/1805.01407.

Broadie, M., Glasserman, P., and Kou, S. (1997). A continuity correction for discrete barrier options. *Mathematical Finance*, 7(4):325–349.

Broadie, M., Glasserman, P., and Kou, S. (1999). Connecting discrete and continuous path-dependent options. *Finance and Stochastics*, 3(1):55–82.

Coskun, S. and Korn, R. (2018). Pricing barrier options in the Heston model using the Heath–Platen estimator. *Monte Carlo Methods and Applications*, 24(1):29–41.

Coskun, S., Korn, R., and Desmettre, S. (2019). Application of the Heath–Platen estimator in the Fong–Vasicek short rate model. *Journal of Computational Finance*.

Daluiso, R. and Facchinetti, G. (2018). Algorithmic Differentiation for Discontinuous Payoffs. *International Journal of Theoretical and Applied Finance*, 21(04):1850019.

Fike, J. and Alonso, J. (2011). The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics.

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*. Springer New York.

Goldman, M. B., Sosin, H. B., and Gatto, M. A. (1979). Path Dependent Options: "Buy at the Low, Sell at the High". *The Journal of Finance*, 34(5):1111.

Gourieroux, C. and Sufana, R. (2004). Derivative pricing with multivariate stochastic volatility: Application to credit risk. Working Papers 2004-31, Center for Research in Economics and Statistics.

Grasselli, M. (2016). The 4/2 stochastic volatility model: A unified approach for the Heston and the 3/2 model. *Mathematical Finance*, 27(4):1013–1034.

Griewank, A. and Walther, A. (2008). *Evaluating Derivatives*. Society for Industrial and Applied Mathematics.

Haug, E. (2007). *The Complete Guide to Option Pricing Formulas*. McGraw-Hill, New York.

Heath, D. and Platen, E. (2002). A variance reduction technique based on integral representations. *Quantitative Finance*, 2(5):362–369.

Heath, D. and Platen, E. (2014). A Monte Carlo Method using PDE Expansions for a Diversifed Equity Index Model. Research Paper Series 350, Quantitative Finance Research Centre, University of Technology, Sydney.

Hellekalek, P. (1998). Don't trust parallel Monte Carlo! In *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233)*, pages 82–89.

Heston, S. L. (1993). A Closed-Form solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *Review of Financial Studies*, 6(2):327–43.

Kloeden, P. E. and Platen, E. (1992). *Numerical Solution of Stochastic Differential Equations (Stochastic Modelling and Applied Probability)*. Springer.

Linetsky, V. (2004). Lookback options and diffusion hitting times: A spectral expansion approach. *Finance and Stochastics*, 8(3):373–398.

Musiela, M. (2005). *Martingale Methods in Financial Modelling*. Springer, Berlin New York.

Øksendal, B. (2003). *Stochastic Differential Equations*. Springer Berlin Heidelberg.

Platen, E. (1997). A non-linear stochastic volatility model. Financial Mathematics Research Reports FMRR 005-97, Australian National University, Canberra.

Platen, E. and Shi, L. (2008). On the Numerical Stability of Simulation Methods for SDES. Research Paper Series 234, Quantitative Finance Research Centre, University of Technology, Sydney.

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-Mode Automatic Differentiation in Julia. *arXiv:1607.07892 [cs.MS]*.

Revuz, D. and Yor, M. (1999). *Continuous Martingales and Brownian Motion*. Springer Berlin Heidelberg.

Claim distribution in continuous time

Johan Auster*

Abstract

This article generalizes the scalable reward distribution methodology (colloquially referred to as "the billion-dollar algorithm") to a continuous time framework encompassing a wide range of special cases. Several smart contract vulnerabilities that arise in practice are discussed in relation to the proposed methods, after which notions are introduced to enable the interpretation of blockchains as stochastic processes compatible with the established framework. Filtrations that capture the sequential flow of information available to smart contracts in their execution are formalized, along with the concept of monitoring of processes that are not observable to contracts in general under these filtrations. Concrete pseudocode implementations are then presented.

Keywords: Claim distribution, constant time algorithms, filtrations, blockchains, smart contracts.

*Department of Mathematical Sciences, University of Copenhagen, Denmark. E-mail: johan.auster@math.ku.dk.

1 Introduction

This article generalizes the scalable reward distribution method (colloquially known as "the billion-dollar algorithm"), originally proposed by Batog et al. (2018) specifically for distribution of rewards on the Ethereum blockchain (Buterin, 2013). A continuous time stochastic setup is formulated to model the *allocation of claims* among a set of *accounts* based on their (changing) *relative shares*, for which a constant time evaluation scheme is derived. Examples of practical special cases with additional structure imposed on claims processes are then covered, including claims *backed* (fully or partially) by deposits, along with *delayed claim allocation* governed by dynamics that derive from these deposits.

Several concepts and definitions are then introduced to enable the discussion of *smart contract* compatible blockchains in a general (implementation-agnostic) way. The presence of multiple simultaneous states in a single block and associated contract vulnerabilities are discussed in relation to the proposed methods. A totally-ordered sequential structure is then proposed, reconciling the simultaneous states in individual blocks with a stochastic process interpretation compatible with the established framework.

Suitable filtrations that model the flow of information within- and across blocks are formalized, which in turn suggest issues in the implicit reliance on deposit processes which are not generally observable by contracts. This is subsequently addressed by the concept of *monitoring* of processes within contract execution contexts, enabling implementations of deposit-based claim allocations based on observable lower bounds.

Special cases of the structure formalized in this article are widely utilized in practice, including in the Uniswap V3 Staker contract.[1] While implementations often rely on instant- or linear allocation of claims to participating accounts, the delayed allocation structure formalized in Section 2.3.3 enables dynamics that derive from a general class of monotonic Lipschitz-continous functions, including the concrete example provided of an exponential specification in Section 4.3.

The article is organized as follows: Section 2.1 provides an illustrative example of the constant time evaluation scheme that is indicative of the more general case to come. Section 2.2 establishes the continuous time stochastic framework for a general class of total claims processes, along with a scheme for the evaluation of claims allocated to individual accounts. Section 2.3 then covers several special cases of this setup.

Section 3.1 develops general notions around blockchains and smart contracts. Section 3.2 discusses the non-uniqueness of state within each block and potential contract vulnerabilities that can arise. Section 3.3 finally provides an interpretation of blockchains as stochastic processes (on a filtered probability space) compatible with the framework- and special cases previously discussed.

Section 4 provides pseudocode of concrete implementations of contracts in the fully backed special case, with one example replicating the instant allocation of claims analogous to Batog et al. (2018), and another demonstrating delayed exponential allocation of claims. Section 5 concludes the article.

[1] `https://web.archive.org/web/20231209115558/https://docs.uniswap.org/contracts/v3/reference/periphery/staker/UniswapV3Staker`

2 Distribution of claims

2.1 An illustrative example

Alice and Bob are co-owners in a new business with an agreement that every owner is entitled to, at any time of their choosing, *redeem* their *claims* to a fraction of total profits in proportion to their relative share in the business at the time of sales. Alice will be responsible for maintaining the accounting of these claims on a *ledger*.

In the first week, the business makes a total profit of \$500 and Bob wants to redeem \$100 of his claims. They have equal shares, and so Alice *allocates* \$250 to herself and $\$250 - \$100 = \$150$ to Bob. Two days later, Alice wants to redeem **all** of her claims; the business has made another \$100, and so Alice adds \$50 to Bob's existing entry of \$150 and clears her own entry to \$0 after receiving her \$300.

Five days later, Alice and Bob agree to each sell 10% of their shares to Charlie. Because the relative stakes are changing, Alice needs to immediately allocate the profits of \$700 accrued since the last update of their spreadsheet: Alice and Bob are allocated \$350 each, followed by adding another entry for Charlie with an initial allocation of \$0. After only thirty minutes, Charlie demands his rightful 20% share of the accumulated \$5 profit - and so each entry is again updated in the ledger: an additional \$2 is allocated to Alice and Bob, while Charlie redeems his \$1 and is left with \$0 in remaining claims.

Five hours later, another \$20 profit has been made, at which point Charlie sells half of his shares to Dan, who plans to gift tiny fractions of his shares to every member of his extended, *liquidity-strapped* family over the coming days. Alice subsequently contemplates whether to hire an accountant or sell her remaining shares in the business.

Alice	Bob	Charlie	Dan	...	**Profit**
0 [50%]	**0** [50%]				0
250 [50%]	**150** [50%]				500
0 [50%]	200 [50%]				100
350 [40%]	**550** [40%]	**0** [20%]			700
352 [40%]	552 [40%]	**0** [20%]			5
360 [40%]	560 [40%]	**4** [10%]	**0** [10%]		20
⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Centralized ledger of claims.

The problem that arises in the above approach is related to the *time complexity* in the structure used for allocating claims: every time **any** owner wants to redeem their claims or transfer their shares, **every** entry of the ledger needs to be updated. More precisely: the number of operations required for updating the ledger grows *linearly*, $O(N)$, in the number of owners N (Knuth, 1997).

Worse still, this linear growth is *per update*. If every new owner redeems claims at similar frequencies independently of other owners, the *frequency* of updates **also** grows linearly. Finally, if new owners continue to sell parts of their own shares at an identical pace to create *even more* owners, we end up with *exponential* growth in the number of

owners, each triggering yet more ledger updates - and with each ledger update requiring a linearly-growing number of operations, ledger management **rapidly** grows out of hand.

Fortunately, an alternative approach exists that requires fewer operations overall **and** distributes the workload among owners in a "fair" way. All owners together keep track of the **total** (cumulative) **profits** noted down on a shared ledger, while each owner additionally tracks two values for themselves: **Redeemable**/**Last Profit**.

Any time any new- or existing owner has a change in their share or wants to redeem their claims, they go through the following steps (in order):

1. Add new profits (if any) to the shared **Total Profit**.
2. Add **Share** $\cdot$ (**Total Profit** $-$ **Last Profit**) to **Redeemable**.
3. Set **Last Profit** equal to **Total Profit**.
4. Subtract redeemed amount (if any) from **Redeemable**.
5. Update **Share** to new value (if any).

Instead of immediately allocating the accumulated profits of each period among all owners according to their relative share, owners *individually* track the accumulation of profits in partitions of time where their own shares remain constant, thereby *decoupling* the timing of allocations from other owners. This leverages that an owners claims within any interval of time with a constant share will be the corresponding share of profits accumulated in this period - independently of how other shares are distributed in the interim.

Unlike the first approach, these steps do not depend **at all** on how many owners have shares in the business, or what other owners do with their shares: the time complexity is *constant* - $O(1)$. No matter how many more owners get involved or how frequently they redeem their share of profits, no other owner has to perform any updates until they themselves redeem claims or have a change in their share.

Alice	Bob	Charlie	Dan	...	**Total Profit**
0/**0** [50%]	**0**/**0** [50%]				0
	150/**500** [50%]				500
0/**600** [50%]					600
350/**1300** [40%]	**550**/**1300** [40%]	**0**/**1300** [20%]			1300
		0/**1305** [20%]			1305
		4/**1325** [10%]	**0**/**1325** [10%]		1325
⋮	⋮	⋮	⋮	⋮	⋮

Table 2: Distributed ledger of claims.

Constant time algorithms are particularly important for *smart contracts* (computer programs that run on a *blockchain*) due to the expensive nature of storage and operations. Methods such as the above are prime candidates in such contexts, as they provide a combination of computational efficiency, scalability and simple handling of transaction costs (with each account simply paying for their own computations and storage, which are unaffected by other accounts).

Smart contracts enable the implementation of methods akin to the above in a *trustless* environment that does not require owners to trust that others accurately and honestly track the redemption of their own claims, as this will be handled autonomously by an immutable set of publicly visible program instructions embedded in smart contracts.

The remainder of the article formalizes the core principle of this approach in the evaluation of allocations from a general continuous time stochastic process to accounts based on their (changing) relative shares. Various aspects of blockchains and smart contracts are then formalized in order to translate these methods into practice, followed by concrete examples of pseudocode implementations.

2.2 Continuous time claim distribution

2.2.1 Share processes

We assume a probability space $(\Omega, \mathcal{F}, P)$ with an augmented filtration $\mathbb{F} \coloneqq (\mathcal{F}_t)_{t\geq 0}$ and let $q \coloneqq (q_t)_{t\geq 0}$ be a left-continuous with right limits (LCRL) process of the form

$$q_t \coloneqq \sum_{i=1}^{\infty} Z_i \mathbf{1}_{\{\tau_{i-1}<t\leq\tau_i\}} \tag{2.1}$$

for an increasing sequence of $\mathbb{F}$-stopping times $\mathcal{Q} \coloneqq (\tau_i)_{i\in\mathbb{N}_0}$ with $0 = \tau_0 < \tau_1 < \dots$ and a sequence of real-valued non-negative random variables $Z \coloneqq (Z_i)_{i\in\mathbb{N}}$ with $Z_1 = 0$, $Z_i \neq Z_{i+1}$ and $\mathcal{F}_{\tau_{i-1}}$-measurability of Z_i for $i \in \mathbb{N}$ such that q is $\mathbb{F}$-adapted; and in particular, $\mathbb{F}$-predictable.

Let $\mathcal{A} \coloneqq \{A, B, \dots\}$ be a countable index set for a family of LCRL processes $p \coloneqq (p^A)_{A\in\mathcal{A}}$ with each process $p^A \coloneqq (p_t^A)_{t\geq 0}$ of the form

$$p_t^A \coloneqq \sum_{i=1}^{\infty} Z_i^A \mathbf{1}_{\{\tau_{i-1}<t\leq\tau_i\}} \tag{2.2}$$

for sequences of real-valued non-negative random variables $Z^A \coloneqq (Z_i^A)_{i\in\mathbb{N}}$ with $\mathcal{F}_{\tau_{i-1}}$-measurability of Z_i^A for $i \in \mathbb{N}$, and assume that $\sum_{A\in\mathcal{A}} Z_i^A \overset{\text{a.s.}}{=} Z_i$ for all $i \in \mathbb{N}$ such that $\sum_{A\in\mathcal{A}} p_t^A \overset{\text{a.s.}}{=} q_t$.

Denote further by $\mathcal{P}^A \coloneqq (\tau_i^A)_{i\in\mathbb{N}_0}$ the smallest subsequences of $\mathcal{Q}$ for which p^A is constant in $(\tau_{i-1}^A, \tau_i^A]$ for $i \in \mathbb{N}$ and let $\bar{Z}^A \coloneqq (\bar{Z}_i^A)_{i\in\mathbb{N}}$ be the corresponding subsequences of Z^A such that $\bar{Z}_i^A \neq \bar{Z}_{i+1}^A$ for $i \in \mathbb{N}$ and

$$p_t^A = \sum_{i=1}^{\infty} Z_i^A \mathbf{1}_{\{\tau_{i-1}<t\leq\tau_i\}} = \sum_{i=1}^{\infty} \bar{Z}_i^A \mathbf{1}_{\{\tau_{i-1}^A<t\leq\tau_i^A\}} \tag{2.3}$$

for all $t \geq 0$ and $A \in \mathcal{A}$.

We refer to p^A as the *shares* of *account* A and define its *relative share* $h^A \coloneqq (h_t^A)_{t\geq 0}$ as

$$h_t^A \coloneqq \begin{cases} \frac{p_t^A}{q_t}, & q_t \neq 0 \\ 0, & q_t = 0 \end{cases}. \tag{2.4}$$

Since $0 \leq p_t^A \leq \sum_{B\in\mathcal{A}} p_t^B \overset{\text{a.s.}}{=} q_t$ for all $t \geq 0$, it follows that $q_t = 0$ implies $p_t^A \overset{\text{a.s.}}{=} 0$ for all

$A \in \mathcal{A}$, while $q_t > 0$ implies

$$\sum_{A\in\mathcal{A}} h_t = \frac{\sum_{A\in\mathcal{A}} p_t^A}{q_t} \overset{\text{a.s.}}{=} \frac{q_t}{q_t} = 1, \tag{2.5}$$

hence the family of relative shares $h \coloneqq (h^A)_{A\in\mathcal{A}}$ almost surely (a.s.) constitute a convex combination on intervals $(\tau_{i-1}, \tau_i]$ where $q_t = Z_i \neq 0$.

2.2.2 Total claims processes

Let the cumulative *total claims* $X \coloneqq (X_t)_{t\geq 0}$ be a real-valued $\mathbb{F}$-adapted right-continuous with left limits (RCLL) process and define the *total allocated claims* as

$$X_t^Q \coloneqq -X_0 \mathbf{1}_{\{t<\tau^+\}} + \sum_{i=1}^{\infty} \left(X_t \mathbf{1}_{\{Z_i>0\}} + X_{\tau_{i-1}} \mathbf{1}_{\{Z_i=0\}} \right) \mathbf{1}_{\{\tau_{i-1}\leq t<\tau_i\}} \tag{2.6}$$

for the first hitting time

$$\tau^+ \coloneqq \inf\{t \geq 0 : q_t > 0\}, \tag{2.7}$$

which is an $\mathbb{F}$-stopping time by the Début theorem (Nikeghbali, 2006). Since $\tau_0 = 0$, $Z_1 = 0$ and $Z_1 \neq Z_2 \geq 0$, it follows that $\tau^+ \overset{\text{a.s.}}{=} \tau_1$.

The total allocated claims process X^Q (2.6) effectively "pauses" the total claims on intervals $[\tau_{i-1}, \tau_i)$ where $q_+ = 0$ (in the sense of right limits), ending in an immediate jump to the value of the underlying total claims process X at τ_i. Subtracting X_0 until the first hitting time of $\{q > 0\}$ further ensures that the total allocated claims start at zero (even if X does not), and remain so up until τ^+.

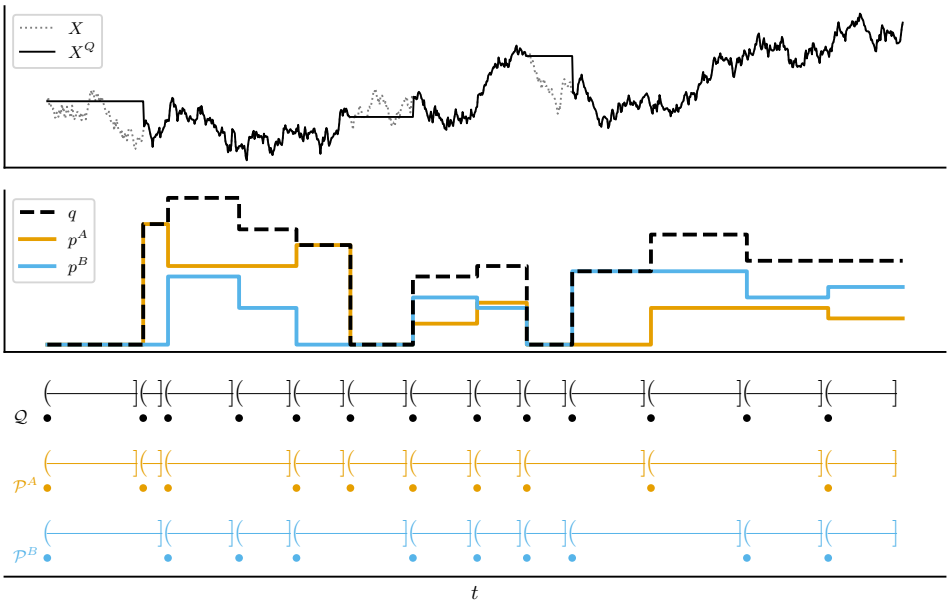


Figure 1: Share processes and allocated total claims X^Q.

We next define the family of $\mathbb{F}$-adapted processes $\Gamma \coloneqq (\Gamma^i)_{i\in\mathbb{N}}$ with $\Gamma^1 = 0$ and

$$\begin{aligned}\Gamma_t^i &\coloneqq \begin{cases} \frac{1}{Z_i}\left(X^Q_{\tau_i\wedge t} - X^Q_{\tau_{i-1}\wedge t} + \Delta X^Q_{\tau_{i-1}}\mathbf{1}_{\{\tau_{i-1}<t\}}\mathbf{1}_{\{Z_{i-1}=0\}}\right), & Z_i > 0 \\ 0, & Z_i = 0 \end{cases} \\ &= \begin{cases} \frac{1}{Z_i}\left(X_{\tau_i\wedge t} - X_{\tau_{i-1}\wedge t} + \Delta X^Q_{\tau_{i-1}}\mathbf{1}_{\{\tau_{i-1}<t\}}\mathbf{1}_{\{Z_{i-1}=0\}}\right), & Z_i > 0 \\ 0, & Z_i = 0 \end{cases}\end{aligned} \tag{2.8}$$

for $i > 1$, where

$$\Delta X_t^Q \coloneqq X_t^Q - \lim_{s\nearrow t} X_s^Q = X_t^Q - X_{t-}^Q \tag{2.9}$$

denotes the jump in X^Q at time t. Each Γ^i process then corresponds to the accumulation of allocated claims normalized by q in a period $(\tau_{i-1}, \tau_i]$ where q is constant, including a (left-continuous) jump term when Z_{i-1} is zero. While each Γ^i is zero up until (and including) time τ_{i-1}, only those *without* the jump term (i.e. with $Z_{i-1} > 0$ or $Z_i = 0$) are also *right-continuous* at τ_{i-1}.

Note that $\Delta X^Q_{\tau_{i-1}}$ for $i > 2$ in (2.8) can be equivalently expressed in terms of the change in X since $X_{\tau_{i-2}}$, as for any $i > 2$ with $Z_{i-1} = 0$ and $Z_i > 0$:

$$\Delta X^Q_{\tau_{i-1}} = X^Q_{\tau_{i-1}} - \lim_{s\nearrow\tau_{i-1}} X_s^Q = X_{\tau_{i-1}} - X_{\tau_{i-2}} \tag{2.10}$$

by (2.6), where $Z_i > 0$ implies $X^Q_{\tau_{i-1}} = X_{\tau_{i-1}}$ and $Z_{i-1} = 0$ implies $X_s^Q = X_{\tau_{i-2}}$ for all $s \in [\tau_{i-2}, \tau_{i-1})$, hence in particular the left limit of τ_{i-1}.

This enables the rewriting of (2.8) for $Z_i > 0$ as

$$\Gamma_t^i = \begin{cases} \frac{1}{Z_i}\left(X_t - X_{\tau_{i-2}}\right), & Z_{i-1} = 0 \\ \frac{1}{Z_i}\left(X_t - X_{\tau_{i-1}}\right), & Z_{i-1} \neq 0 \end{cases} \tag{2.11}$$

for $i > 2$ with $t \in [\tau_{i-1}, \tau_i]$, from which it follows for $s \in [\tau_{i-1}, t)$ that

$$\begin{aligned}\Gamma_t^i - \Gamma_s^i &= \begin{cases} \frac{1}{Z_i}\left(X_t - X_{\tau_{i-2}}\right) - \frac{1}{Z_i}\left(X_s - X_{\tau_{i-2}}\right), & Z_{i-1} = 0 \\ \frac{1}{Z_i}\left(X_t - X_{\tau_{i-1}}\right) - \frac{1}{Z_i}\left(X_s - X_{\tau_{i-1}}\right), & Z_{i-1} \neq 0 \end{cases} \\ &= \frac{1}{Z_i}\left(X_t - X_s\right).\end{aligned} \tag{2.12}$$

Meanwhile for $i = 2$ with $\tau_{i-1} = \tau_1 \overset{\text{a.s.}}{=} \tau^+$:

$$\lim_{s\nearrow\tau_1} X_s^Q \overset{\text{a.s.}}{=} \lim_{s\nearrow\tau^+} X_s^Q = -X_0 + X_{\tau_0} = -X_0 + X_0 = 0 \tag{2.13}$$

by construction (2.6) such that $\Delta X^Q_{\tau^1} \overset{\text{a.s.}}{=} X_{\tau_1}$ and

$$\Gamma_t^2 = \frac{1}{Z_2}\left(X_t^Q - X^Q_{\tau_1} + \Delta X^Q_{\tau^1}\right) \overset{\text{a.s.}}{=} \frac{1}{Z_2}X_t^Q = \frac{1}{Z_2}\left(X_t - 0\right) \tag{2.14}$$

for $t \in [\tau_1, \tau_2]$.

The above shows how both the direct evaluation of Γ^i on $[\tau_{i-1}, \tau_i]$ and incrementing

this from a previous value of Γ^i have the same form: namely as the change since some previously observed value of X (starting at zero) up to the "current time" scaled by $1/z_i$ - which is not surprising, since all Γ^i start at zero.

We finally define the *total relative claims* as

$$\xi_t \coloneqq \sum_{i=1}^{\infty} \Gamma_t^i \tag{2.15}$$

which will represent the cumulative allocated claims *per share*. Since each Γ^i in this sum will be constant outside "its interval" $(\tau_{i-1}, \tau_i]$, an incremental scheme for evaluating (2.15) up to an arbitrary time $t \geq 0$ arises in a natural way from the structure on the individual Γ^i processes noted above.

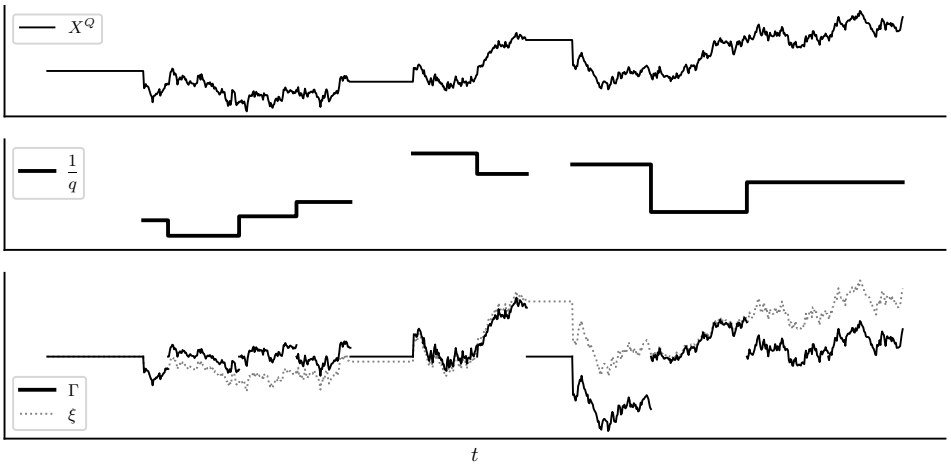


Figure 2: Total relative claims $\xi = \sum_{i=1}^{\infty} \Gamma^i$ and inverse total shares $1/q$.

2.2.3 Incremental evaluation of relative claims

We now show how the total relative claims ξ (2.15) can be evaluated in an incremental manner, keeping track of only the *running* value of ξ, previous values of X and (right limits) q_+. By construction we have that $\xi_0 = 0$, and this remains so at least up until time $\tau^+ \overset{\text{a.s.}}{=} \tau_1$, at which point the value $q_{\tau_1+} = Z_2 > 0$ is stored as the *variable* q.

Following this, ξ can (optionally) be incremented to any arbitrary time $s \in (\tau_1, \tau_2)$ using

$$\xi_s = \underbrace{\Gamma_s^1}_{=0} + \Gamma_s^2 + \sum_{i=3}^{\infty} \underbrace{\Gamma_s^i}_{=0} = \frac{1}{Z_2} X_s^Q = \frac{1}{q} X_s^Q, \tag{2.16}$$

at which point the value of X_s is stored as X_-, allowing for further incrementing ξ by

$$\xi_t - \xi_s = \left(\Gamma_t^2 - \Gamma_s^2\right) = \frac{1}{q}\left(X_t - X_-\right) \tag{2.17}$$

up to any time $t < \tau_2$, continuously updating the stored value of X_- on each increment.

From time τ_2, Γ^2 will remain constant up to all future times, and so the final value of Γ^2 is incorporated into ξ, with the values X_{τ_2} and $q_{\tau_2+} = Z_3$ stored for subsequent increments

via evaluations of Γ^3. Notably, this does not require having evaluated Γ^2 at any previous point of the interval due to the aforementioned equivalence between incrementing Γ^2 and direct evaluation from the stored X_- value: adding the change in X scaled by $1/q$ will always ensure that Γ^2 is fully captured in ξ.

This procedure can be applied indefinitely moving forward, updating the values of X_- (and q) whenever ξ is (optionally) evaluated between- or *at* times (necessarily) where q has a discontinuity. When times τ_{i-1} with $Z_i = 0$ are reached, $X_{\tau_{i-1}}$ is kept until τ_i with $Z_{i+1} > 0$, at which point q is updated and the steps outlined above are repeated.

Algorithm 1 Evaluation of ξ_{t_N} for a supersequence $0 = t_0 < t_1 < \ldots < t_N$ of $\mathcal{Q}$.

Input: $(X_{t_0}, X_{t_1}, \ldots, X_{t_N})$, $(q_{t_0+}, q_{t_1+}, \ldots, q_{t_N+})$
Output: ξ_{t_N}

$\xi, X_-, q \leftarrow 0$ ▷ *Initialize variables*
for $n = 0, 1, \ldots, N$ **do**
  if $q > 0$ **then**
    $\xi \leftarrow \xi + \frac{1}{q}(X_{t_n} - X_-)$ ▷ *Increment ξ based on change in X*
    $X_- \leftarrow X_{t_n}$ ▷ *Store X_{t_n} for next change in X*
  $q \leftarrow q_{t_n+}$ ▷ *Update q for next ξ increment*
return ξ

Despite the slightly involved nature of constructing the underlying processes, the practical evaluation as outlined in the pseudocode of Algorithm 1 is very simple. The same update logic is applied at all times, and all of the above cases are captured by simply updating q *after* incrementing ξ and updating X_- (whenever the previous value of q was positive), with all variables initialized to zero. As long as *at least* all stopping times $\tau_i, i \in \mathbb{N}_0$ in $\mathcal{Q}$ where q is discontinuous are included in the evaluation, this scheme will track ξ **exactly**.

2.2.4 Account claim allocations

We let X^A refer to the cumulative claims allocated to account $A \in \mathcal{A}$ from X^Q based on its relative share h^A such that for $\tau_{i-1} < s \leq t \leq \tau_i$:

$$X_t^A - X_s^A = h_s^A \left(X_t^Q - X_s^Q \right) = \frac{Z_i^A}{Z_i} \left(X_t^Q - X_s^Q \right). \tag{2.18}$$

It may then be tempting to take X^A to be the stochastic integral $\int h^A \, \mathrm{d}X^Q$ (Revuz and Yor, 1999) - however this has the problem that for e.g. $t \in (\tau_1, \tau_2]$ where $\tau_1 \overset{\text{a.s.}}{=} \tau^+$:

$$\begin{aligned}\sum_{A\in\mathcal{A}}\int_0^t h_u^A \,\mathrm{d}X_u^Q &= \sum_{A\in\mathcal{A}}\left(\int_0^{\tau^+} \underbrace{h_u^A}_{=0}\,\mathrm{d}X_u^Q + \int_{\tau^+}^t h_u^A\,\mathrm{d}X_u^Q\right)\\ &= \sum_{A\in\mathcal{A}}\left(\frac{Z_2^A}{Z_2}\left(X_t^Q - X_{\tau^+}^Q\right)\right)\\ &= \left(X_t^Q - X_{\tau^+}^Q\right)\sum_{A\in\mathcal{A}}\frac{Z_2^A}{Z_2}\\ &= X_t^Q - X_{\tau^+}^Q\\ &\neq X_t^Q\end{aligned} \tag{2.19}$$

such that any jumps in X^Q at τ^+ that were explicitly accounted for in the Γ processes (2.8) are missed, hence the sum of these integrals will **not** in general correspond to the total allocated claims X^Q.

Instead, we take

$$X_t^A \coloneqq \sum_{i=1}^{\infty} Z_i^A\left(\xi_{\tau_i\wedge t} - \xi_{\tau_{i-1}\wedge t}\right) = \sum_{i=1}^{\infty} Z_i^A\Gamma_t^i \tag{2.20}$$

with the latter equality following from

$$\xi_{\tau_i\wedge t} - \xi_{\tau_{i-1}\wedge t} = \sum_{j=1}^{\infty}\Gamma^j_{\tau_i\wedge t} - \sum_{j=1}^{\infty}\Gamma^j_{\tau_{i-1}\wedge t} = \Gamma^i_{\tau_i\wedge t} - \underbrace{\Gamma^i_{\tau_{i-1}\wedge t}}_{=0} = \Gamma_t^i, \tag{2.21}$$

since the Γ^j terms outside $j=i$ cancel, Γ^i is zero up until (and including) τ_{i-1} and Γ^i is constant after τ_i. The account claims X^A will then correspond to the Γ processes scaled by the shares of account A in the applicable intervals, and with $t\in(\tau_1,\tau_2]$ as above:

$$\begin{aligned}\sum_{A\in\mathcal{A}} X_t^A &= \sum_{A\in\mathcal{A}}\left(\sum_{j=1}^{\infty} Z_j^A\Gamma_t^j\right)\\ &= \sum_{A\in\mathcal{A}}\left(Z_1^A\underbrace{\Gamma_t^1}_{=0} + Z_2^A\Gamma_t^2 + \sum_{j=3}^{\infty} Z_j^A\underbrace{\Gamma_t^j}_{=0}\right)\\ &\overset{\text{a.s.}}{=} \sum_{A\in\mathcal{A}}\left(Z_2^A\frac{1}{Z_2}\left(X_t^Q - X_{\tau^+}^Q + \Delta X_{\tau^+}^Q\right)\right)\\ &= X_t^Q\underbrace{\sum_{A\in\mathcal{A}}\left(Z_2^A\frac{1}{Z_2}\right)}_{=1}\end{aligned} \tag{2.22}$$

since the jump in X^Q at time $\tau^+ \overset{\text{a.s.}}{=} \tau_1$ is included in the Γ^2 process by (2.8). Further increments will then simply be the sum of the scaled changes in X up until the next time with $q_+ = 0$, after which the jump will again be incorporated in a subsequent Γ process when $q > 0$.

The Γ processes are constructed to explicitly handle the issue of left-continuous integrands h^A with respect to an RCLL integrator X^Q potentially "missing" the point mass of X^Q at the onset of intervals with $h^A_+ > 0$ following $h^A = 0$. By *shifting* the jump term into a left-continuous term in ξ, the account claims processes as defined in (2.20) **will** include these jumps.

In general, the sum over all accounts will agree with X^Q at all points **except** the singular points in time τ_i at which X^Q jumps in the transition from an interval $[\tau_{i-1}, \tau_i,)$ with $Z_i = 0$ to $[\tau_i, \tau_{i+1})$ with $Z_{i+1} > 0$ due to the left-continuity of the jump term in (2.8).

2.2.5 Incremental evaluation of account claims

We finally show how the evaluation of account claims can be combined with the evaluation of the total relative claims ξ discussed in Section 2.2.3 in a manner that *distributes* the computation of increments of ξ across accounts, analogous to the illustrative example covered in Section 2.1.

First, Algorithm 2 shows the pseudocode for the evaluation of some isolated X^A up to time t_N based on the definition in (2.20) using the previous steps for evaluating ξ. This requires additional variables for keeping track of X^A and the changes in ξ between increments. The evaluation necessitates incrementing ξ *at least* at all times $\tau_0, \tau_1, \ldots$ where q is discontinuous - even across intervals of times in which p^A remains constant throughout.

Algorithm 2 Evaluation of $X^A_{t_N}$ for a supersequence $0 = t_0 < t_1 < \ldots < t_N$ of $\mathcal{Q}$.

Input: $(X_{t_0}, X_{t_1}, \ldots, X_{t_N})$, $\left(p^A_{t_0+}, p^A_{t_1+}, \ldots, p^A_{t_N+}\right)$, $(q_{t_0+}, q_{t_1+}, \ldots, q_{t_N+})$
Output: $X^A_{t_N}$

$\xi, \xi^A_-, X_-, X^A, p^A, q \leftarrow 0$ ▷ *Initialize variables*
for $n = 0, 1, \ldots, N$ **do**
  if $q > 0$ **then**
    $\xi \leftarrow \xi + \frac{1}{q}\left(X_{t_n} - X_-\right)$ ▷ *Increment ξ based on change in X*
    $X_- \leftarrow X_{t_n}$ ▷ *Store X_{t_n} for next change in X*
  $X^A \leftarrow X^A + p^A\left(\xi - \xi^A_-\right)$ ▷ *Increment X^A account claims*
  $\xi^A_- \leftarrow \xi$ ▷ *Store ξ for next increment of account claims*
  $q \leftarrow q_{t_n+}$ ▷ *Update q for next ξ increment*
  $p^A \leftarrow p^A_{t_n+}$ ▷ *Update p^A for next X^A increment*
return X^A

However, for any interval $(s, t] \subseteq (\tau^A_{i-1}, \tau^A_i]$ with τ^A and $\bar{Z}^A$ as defined in Section 2.2.1:

$$X^A_t - X^A_s = \bar{Z}^A_i\left(\xi_t - \xi_s\right) \tag{2.23}$$

since $p^A = \bar{Z}^A_i$ on $(\tau^A_{i-1}, \tau^A_i]$, i.e. only the values ξ_t and ξ_s are needed in addition to $\bar{Z}^A_i$ to evaluate the increment in X^A in intervals where p^A is constant - even if **many** discontinuities in q occurred in $(s, t]$.

Since ξ is identical across X^A for all $A \in \mathcal{A}$ and no discontinuities in **any** p^A can occur outside of times $\tau_0, \tau_1, \ldots$ in $\mathcal{Q}$, the incremental updates of ξ needed for evaluating X^A when p^A has a discontinuity can be *distributed* across accounts. By incrementing a shared *global* ξ value each time **any** account has a change in shares (and thus would need to

increment ξ regardless in order to then increment X^A prior to a change in shares), all accounts are ensured an up-to-date ξ value "for free".

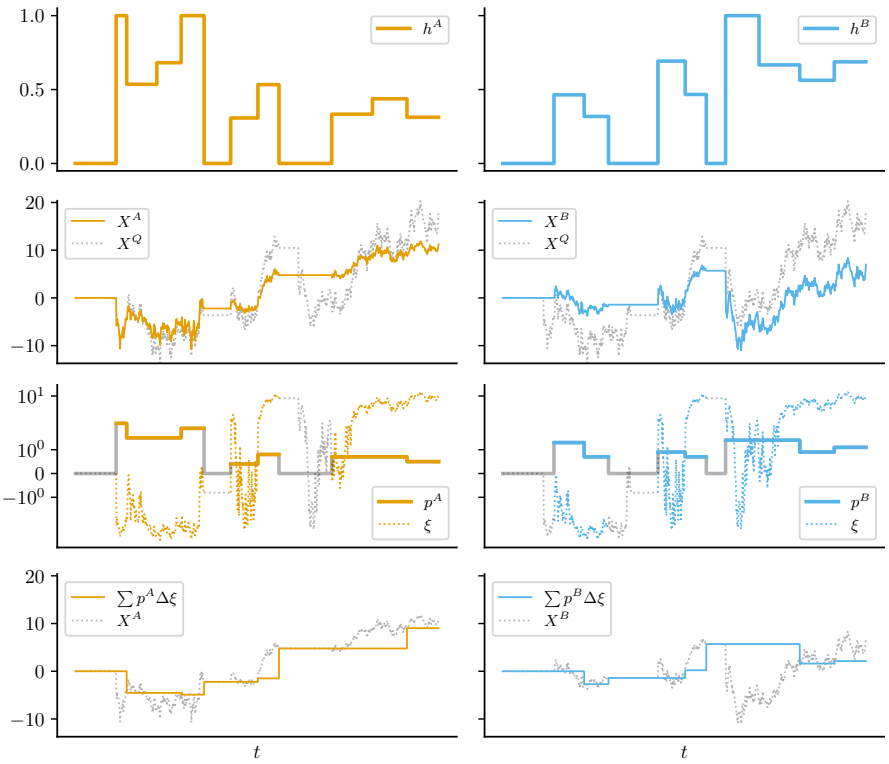


Figure 3: Account claims X^A as sums of $p^A \Delta\xi$ over $\mathcal{P}^A$.

Sharing the incremental computation of ξ enables a substantial reduction in the steps necessary for evaluating X^A in cases where p^A has few changes relative to q, as incrementing ξ will then only be necessary at times in $\mathcal{P}^A$ (where p^A is discontinuous), i.e. from the perspective of account A, $t_0 < t_1 < \ldots < t_N$ need only be a supersequence of $\mathcal{P}^A$ - not $\mathcal{Q}$. This would even work for Algorithm 2 without any further changes - other than making ξ, X_- and q global variables shared across accounts with the assumption that these are updated by *at least* one account at all times $\tau_0 < \tau_1 < \ldots$ in $\mathcal{Q}$.

Algorithm 2 remains *relatively* well-behaved in this setup even when multiple X^A are evaluated at the same time t - because X is global and updated to X_t on the first increment to ξ, any subsequent increments at time t will necessarily be zero, since the stored value of X will already be X_t, and the initial increment will always use the correct value of q, as this cannot be updated before the ξ increment.

However, with *multiple* executions of Algorithm 2 at time t, it is possible for the stored value of q to change from $q = 0$ initially to $q > 0$ after the first execution, leading to the jump term in (2.8) potentially being incorporated in a right-continuous manner. This can be prevented by (globally) storing the time of updates to q and explicitly skipping the ξ update step whenever the current time is not (strictly) greater, as shown in Algorithm 3, ensuring that ξ is only updated *after* $q_+ = 0$ - consistent with the left-continuity of the jump term in Γ and that these always start at zero.

In practice, the value of q may not itself be readily available without each individual change in shares being "registered" - and in fact, this too can be tracked as part of the

distributed scheme as long as the above prevention of multiple updates of ξ at the same time $t \geq 0$ is implemented. Whenever an account has a change in their share, they simply increment both their (local) stored p^A variable and the (global) total shares q. Since multiple increments of ξ cannot happen at time t, the *order* in which these changes in shares are added to q does not matter, as any allocation of claims will be delayed until *after* time t (where all changes in shares will have already been incorporated into q).

Algorithm 3 Distributed evaluation of $X^A_{t_N}$ for a supersequence $0 = t_0 < t_1 < \ldots < t_N$ of $\mathcal{P}^A$. Assumes global variables are kept up-to-date via increments from other accounts at the time of changes in their share.

Input: $(X_{t_0}, X_{t_1}, \ldots, X_{t_N})$, $\left(\Delta p^A_{t_0+}, \Delta p^A_{t_1+}, \ldots, \Delta p^A_{t_N+}\right)$
Output: $X^A_{t_N}$

Globals: $\xi, X_-, q, t_- \leftarrow 0$ ▷ *Initialize global variables*
Locals: $\xi^A_-, X^A, p^A \leftarrow 0$ ▷ *Initialize local (per account) variables*
for $n = 0, 1, \ldots, N$ **do**
  if $q > 0 \wedge t_n > t_-$ **then**
    $\xi \leftarrow \xi + \frac{1}{q}(X_{t_n} - X_-)$ ▷ *Increment ξ based on change in X*
    $X_- \leftarrow X_{t_n}$ ▷ *Store X_{t_n} for next change in X*
  $X^A \leftarrow X^A + p^A\left(\xi - \xi^A_-\right)$ ▷ *Increment X^A account claims*
  $\xi^A_- \leftarrow \xi$ ▷ *Store ξ for next increment of account claims*
  $q \leftarrow q + \Delta p^A_{t_n+}$ ▷ *Increment q for next ξ increment*
  $p^A \leftarrow p^A + \Delta p^A_{t_n+}$ ▷ *Increment p^A for next X^A increment*
  $t_- \leftarrow t$ ▷ *Store current time to prevent simultaneous updates*
return X^A

2.3 Special cases

2.3.1 Bounded claims

Until now, we worked with very general total claims processes X, imposing only that these were RCLL and $\mathbb{F}$-adapted. Notably, X in this setup can decrease and even be *negative*, in which case a likely interpretation of *redeeming* a claim would be the account *paying out* the (absolute) allocated amount, rather than receiving it.

In this and subsequent sections, we focus on special cases with non-negative *bounded claims*, restricting the total claims process X further to be non-negative, non-decreasing (hence finite variation) and bounded from above by a process $U \coloneqq (U_t)_{t\geq 0}$, assumed to also be a non-negative non-decreasing $\mathbb{F}$-adapted real-valued RCLL process such that

$$X_t \leq U_t \tag{2.24}$$

a.s. at all times $t \geq 0$. Note that if this bound is satisfied for X, the allocated process X^Q (see Section 2.2.2) will **also** adhere to this upper bound, since $X^Q \leq X$ if X is non-negative and non-decreasing.

By requiring claims to be non-negative and non-decreasing, we exclude the potential for redemption of claims to have a *timing* component to consider. For example, if redemption of allocated claims by accounts is voluntary, accounts $A \in \mathcal{A}$ are incentivized to time their redemption of claims when the value of X^A is "high" - and accounts may **never** redeem

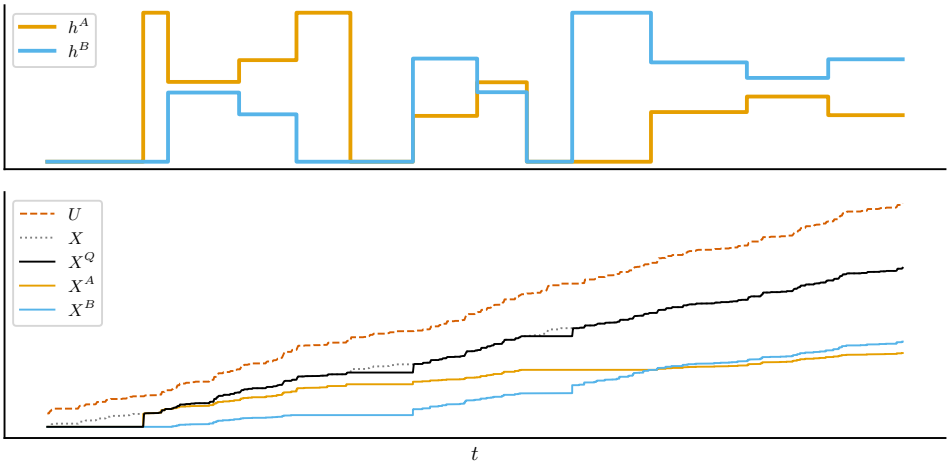


Figure 4: Claims processes bounded by U.

a negative claim. If claims are non-decreasing, there is no such timing factor in the redemption, as allocations can only increase over time.

2.3.2 Fully backed claims

Take two processes Y^+, Y^-, both assumed to be non-negative non-decreasing $\mathbb{F}$-adapted real-valued RCLL processes with $Y^+ \geq Y^-$, and let

$$Y_t := Y_t^+ - Y_t^- \tag{2.25}$$

represent the *net asset balance process* of some account, composed of subtracting *cumulative withdrawals* (outflows) Y^- from *cumulative deposits* (inflows) Y^+.

If we interpret the account balance of Y as one *backing* the total claims X such that Y^- corresponds precisely to the *redemption* of claims (hence $Y^- \leq X^Q \leq X$), we can choose $U := Y^+$ as an upper bound in (2.24) to ensure that any allocated claims are *fully backed* by a corresponding deposit represented in the underlying account for Y (which we refer to as the *claims account*), since then

$$Y_t = Y_t^+ - Y_t^- \geq Y_t^+ - X_t = U_t - X_t \geq 0 \tag{2.26}$$

a.s. at all times $t \geq 0$ - i.e. the net balance process remains non-negative, hence the claims account will always be solvent with respect to the redemption of claims.

Taking then e.g. $X := Y^+$ will correspond to the instant allocation of new deposits towards claims across accounts with positive shares p, as in the Batog et al. (2018) paper (see Section 4.2). Note that Y may itself be derived from some other account - e.g. as some fraction of inflows to a revenue account.

2.3.3 Delayed claim allocation from deposits

Take a real-valued function $f(t,x) : [0,\infty)^2 \to [0,\infty)$ continuous in t, Lipschitz-continuous in x and non-decreasing in x (for fixed t) with

$$f(t,0) = 0 \tag{2.27}$$

for all $t \geq 0$.

We then define R as the finite variation process governed by the stochastic differential equation (SDE)

$$\mathrm{d}R_t = -f(t, R_{t_-})\,\mathrm{d}t + \mathrm{d}Y_t^+ \tag{2.28}$$

with $R_0 = Y_0^+$, where t_- denotes the left limit such that R_{t_-} is left-continuous, hence predictable. Continuity assumptions on f then ensure the existence and uniqueness of solutions (Kloeden and Platen, 1992).

Since f is non-negative, it follows that

$$R_t = Y_0^+ + \int_0^t \mathrm{d}Y_s^+ - \int_0^t f(s, R_{s_-})\,\mathrm{d}t = Y_t^+ - \int_0^t f(s, R_{s_-})\,\mathrm{d}t \leq Y_t^+. \tag{2.29}$$

As Y^+ is also non-negative and the integral of the drift term is continuous, (2.27) together with $R_0 \geq 0$ implies that R is well-defined and non-negative: the drift may "push down" the value of R continuously (which starts from a non-negative initial value) until it reaches zero - but not further.

Taking then

$$X_t \coloneqq Y_t^+ - R_t \tag{2.30}$$

leads to claims which are non-negative, non-decreasing and adhering to the upper bound $X \leq Y^+$, ensuring solvency of the underlying account, and with the immediate jumps in Y^+ cancelled out by the definition of R. This gives an interpretation of R as the *remaining deposits* of the total cumulative deposits not yet allocated (hence *delayed*) towards claims, with the drift coefficient $-f(t, R_{t_-})$ controlling the rate of allocation.

If we further assume that Y^+ is a (pure) jump process, hence a.s. constant on countably many intervals, the SDE (2.28) on any such interval $[a, b)$ with $\mathrm{d}Y^+ = 0$ reduces to an ordinary differential equation (ODE):

$$\mathrm{d}y_t = -f(t, y_t)\,\mathrm{d}t \tag{2.31}$$

with initial condition $y_0 = R_a$, which for e.g. $f(t, x) = rx$ with $r \geq 0$ has the solution

$$R_s = R_a e^{-r(s-a)} \tag{2.32}$$

for all $s \in [a, b)$. A concrete example of a contract implementation utilizing this specification is covered in Section 4.3.

The standard ODE dynamics in (2.31) are particularly relevant due to the *monitoring* of deposit processes in practice, which by design incorporates jumps in Y^+ **after** evaluating the ODE dynamics based on the last observed value of Y^+, as discussed further in Section 3.3.5. When analytical solutions to the ODE (2.31) are available, the implementation of delayed claim allocations is then as simple as incrementing ξ based on the X resulting from directly evaluating these solutions.

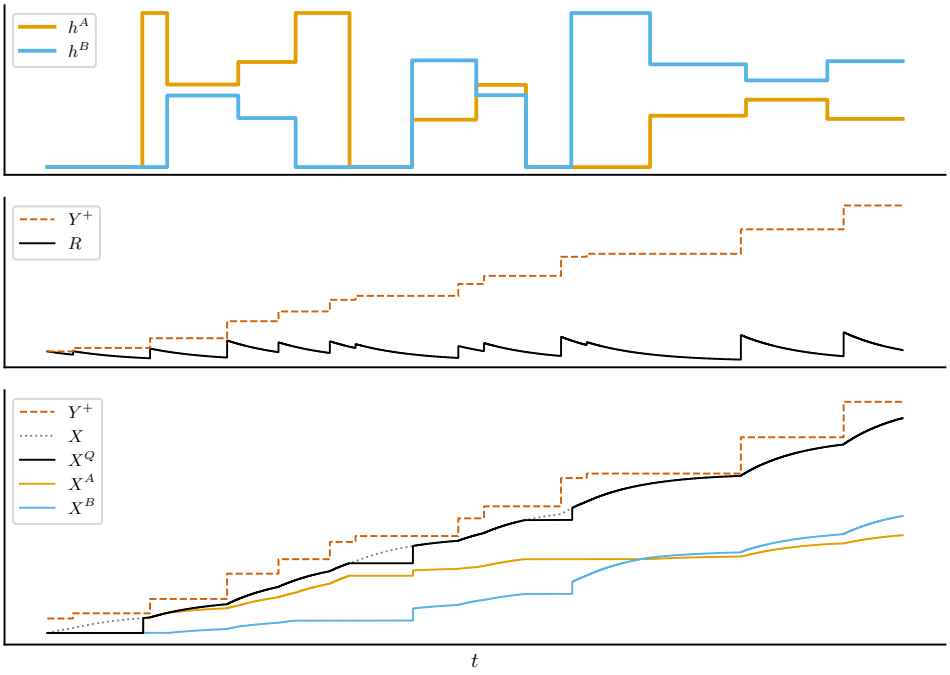


Figure 5: Delayed allocation of claims with drift $f(t,x) \coloneqq rx$ for $r > 0$.

2.3.4 Partially backed claims

An immediate concern regarding the "fully backed" setup in light of the delayed allocation approach is *capital efficiency*: the unallocated deposits $Y^+ - X$ are effectively unused capital, serving no immediate purpose other than to ensure solvency for redemption of claims in the (possibly very distant) future. This motivates another interpretation of Y^+ as still governing the dynamics of X via (2.28), but without necessarily requiring the claims account to *always* remain solvent for **all** future claims.

One way to express this is with the addition of a *net outstanding balance*

$$L_t \coloneqq L_t^+ - L_t^- \tag{2.33}$$

with L^+, L^- analogous to Y^+, Y^- in Section 2.3.2 and $L \leq Y$ a.s..

This can then represent inflows and outflows "owed" to the claims account, which are to be ignored for the purposes of the (delayed) allocation of claims - i.e. not accounted for in Y^+. An increase in L^+ would be a withdrawal *from* the claims account, while an increase in L^- would be a repayment *into* the claims account, and $L \overset{\text{a.s.}}{=} 0$ at all times would correspond to the fully backed case.

Since $Y^+ - X$ are the not-yet-allocated deposits and L represents the amount "missing" from the claims account, it follows that

$$L_t > Y_t^+ - X_t \tag{2.34}$$

implies insolvency at time t, with $L - (Y^+ - X)$ corresponding to the excess amount withdrawn from the claims account. A sensible (bare minimum) control to impose is thus the rejection of any attempted withdrawals that would cause L to exceed this limit, ensuring that any already-allocated claims remain redeemable.

As an example of handling insolvency events, one could, prior to updating ξ as in Algorithm 3, check if the computed value of X from the delayed claim dynamics satisfies (2.34). If this is **not** the case, the claims account is solvent and the scheme can proceed as normal. Otherwise, the change in total claims used in the update of ξ may be replaced by $Y^+ - L$, i.e. the remaining deposits (effectively stopping X after crossing $Y^+ - L$).

This distributes all remaining deposits in the claims account according to the relative shares at the time of the detected insolvency event *independently* of the timing of claim redemptions (preventing a "run on the bank" where accounts rush to redeem their claims prior to insolvency). This may be followed by additional logic, such as terminating all future claim allocations and enabling the seizure of some collateral in proportion to the relative shares of each account.

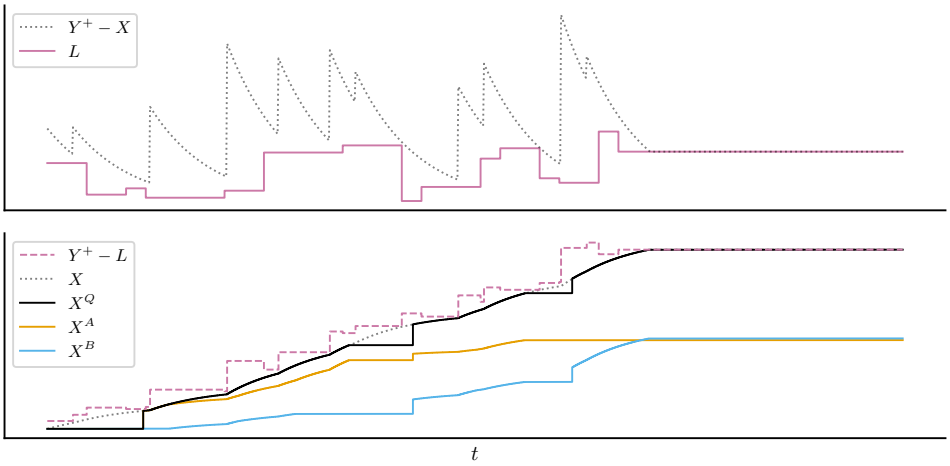


Figure 6: Partially backed claims terminating at insolvency $L > Y^+ - X$.

A myriad of variations around this setup can be devised, including complex logic triggered by insolvency events (e.g. auctioning off illiquid collateral), claim allocation itself being contingent on L (e.g. rate increases as a "risk premium" for insolvency risk), further limitations on withdrawals to provide a guaranteed buffer period of solvency and so forth. Further exploration of such extensions are out of scope of the article and we conclude the theoretical treatment of the claims distribution mechanisms here.

In the coming sections, we formalize some general notions around blockchains in order to interpret and implement the established methods in such environments (where a single point in time is mapped to *multiple* states). A common vulnerability to transaction reordering and how this relates to contracts that implement claim distribution is discussed, followed by the formalization of a stochastic process interpretation of blockchains and associated filtrations. The concept of *monitoring* of cumulative deposits is then introduced in order to derive processes that will be adapted to the filtrations that arise in the execution of contracts.

3 Blockchains as stochastic processes

3.1 Blockchains and contracts

We define a *blockchain* as a sequence of *blocks* $(B_n)_{n\in\mathbb{N}_0}$, with each element of a block mapping to a *block state space* S (containing e.g. all possible timestamps, configurations of assets, account balances, contracts etc.) starting from some initial block $n = 0$ with *block time* $t_n \geq 0$ representing the *physical time* of the block.

Each block $B_n = ((B_n)_m)_{m\in\mathbb{N}_0}$ itself consists of a sequence of *executed instructions* with $m_n + 1 \in \mathbb{N}$ of these non-empty, each of which may mutate the state of the block from the previous instruction. For example, an asset transfer of \$5 from account A to account B can be considered an instruction that decreases the account A balance by \$5 and increases the account B balance by \$5. The state at the *block header* instruction $m = 0$ is assumed to be the state from the last non-empty instruction at m_{n-1} of the previous block after updating the block time to $t_n > t_{n-1}$.[2]

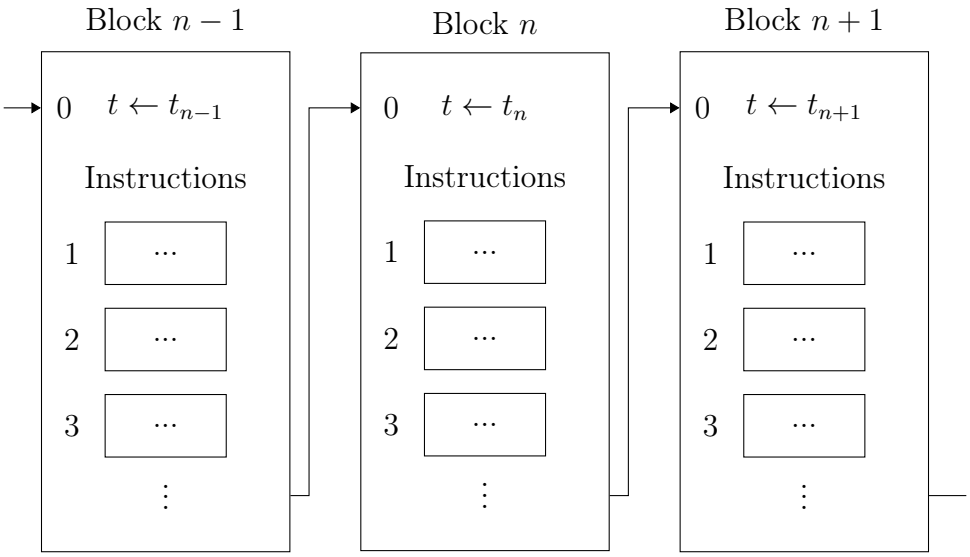


Figure 7: Sequences of blocks (of sequences of instructions).

An *account* A represents an autonomous entity with the capability to create new assets, transfer assets to other accounts, deploy new contracts to the blockchain or interact with existing contracts. A *contract* C is itself an entity *deployed* to the blockchain by an account, embedded with its own *contract account*, *contract storage* and a *contract program* $\Pi^C := (\pi_0, \pi_1, \ldots, \pi_{m_C})$ forming a sequence of $m_C + 1 \in \mathbb{N}$ *program instructions* - always ending in a `return` instruction to end its execution. The program controls the account- and storage space of the contract, and may be *called* by an account, triggering the execution of the program (in accordance with parameters provided in the contract call) - possibly with some temporary limited control over the calling account.

Contracts will often access some element of the state of the block within their program, such as the asset balance of a particular account A, the state of contract storage (possibly of another contract) or even the program instructions of another deployed contract. All of

[2]Typically the block header contains a variety of other meta data regarding the block (though this is less relevant for our purposes and therefore left out of the discussion).

these serve as examples of elements contained within a general block state in S, assumed to be *observable* to contracts at the time of their execution.[3]

We refer to (possibly singular) adjacent related instructions in sequential order which **cannot** be internally reordered as *transactions*, forming a partition of all non-empty instructions in a block. As an example, a *single* transaction can consist of an instruction of account A transfering an asset amount to the contract account of C, immediately followed by calling the program of C (resulting in further executed instructions produced by the contract program). This is **much** broader than the traditional use of the word and can include several complex contract calls within a single transaction.

In very general terms, the blocks of executed instructions arise from accounts "submitting" transactions - within relatively short intervals, these batches of transactions are grouped together and *validated* by *block validators* (sometimes called *miners*), who assemble, verify and execute these transactions in some sequential order, with the result of these forming a new block.

It is important to distinguish between **executed** instructions, which are "actions taken" within a block (akin to running a computer program), and **program** instructions, which are the elements forming the logic of a contract program (akin to the source code of a computer program). We always take "instruction" to implicitly refer to an executed instruction within a block, and use program instruction explicitly whenever referring to parts of a contract program.

A program instruction may be a simple asset transfer from a contract account - but can also be analogous to those of a general-purpose computer. Program instructions in Π^C might increment a value in storage, load the value of an element from the current block state, check that some condition on this value is met and branch to another program instruction based on this. These are examples of program instructions, and calling the contract produces a sequence of executed instructions based on these, immediately following the contract call instruction in the block.

For our purposes, we assume an *interpreted* setting in the execution of programs, in the sense that any program instruction that is "reached" in the execution of a program will correspond to exactly one executed instruction in the resulting block (even if this does not mutate the block state). All executed instructions thus *originate* from (are the execution of the program instruction of) **exactly** one program index $i \in \{0, 1, \dots, m_C\} =: \mathcal{I}^C$.[4]

We say the i'th (executed) instruction of block n is in the k'th *execution context* $\mathcal{E}_k^C$ of contract C when the instruction originates from **any** program index $i \in \mathcal{I}^C$ of the program in the k'th call to the contract, and we let

$$\mathcal{M}_{n,m}^C \coloneqq \left\{ j \in \{0, 1, \dots, m-1\} : (B_n)_{j+1} \in \mathcal{E}_k^C \text{ for some } k \in \mathbb{N}_0 \right\} \tag{3.1}$$

be the set of instruction indices $j < m$ of the n'th block *immediately preceding* instructions

[3] In practice, this access would typically happen via dedicated *read instructions*, loading the relevant values into the *stack* - this level of detail is out of scope of the current discussion, but is explored further in the Ethereum papers by Buterin (2013) and Wood (2014).

[4] An executed instruction on a block does not necessarily originate from a program instruction (e.g. an asset transfer directly from one account to another). Meanwhile, a contract call does not necessarily reach every index of a program, since program instructions may branch to other parts of the program - and finally, the same index may be reached **many** times in a single call, for example in a `for` loop.

in any execution context of C. This then represents the indices of block state that can be accessed in the execution of individual program instructions up to instruction m.

We further take

$$\bar{\mathcal{E}}^C_{n,m} \coloneqq \left\{\mathcal{E}^C_k : (B_n)_j \in \mathcal{E}^C_k \text{ for some } k \in \mathbb{N}_0 \text{ and } j \leq m\right\} \tag{3.2}$$

as all execution contexts that have been *reached* up to (and including) the m'th instruction of block n.

Finally we let

$$\bar{\mathcal{M}}^C_{n,m} \coloneqq \mathcal{M}^C_{n,m} \cup \left\{j \in \mathbb{N}_0 : (B_n)_j \in E \text{ for some } E \in \bar{\mathcal{E}}^C_{n,m}\right\} \tag{3.3}$$

contain all indices of instructions that precede or are **within** execution contexts that have been reached by the m'th instruction. This is then "forward-looking" in the sense that indices greater than m may be included, which will be useful in characterizing the block state observable to a contract *overall*, as discussed further in Section 3.3.4.

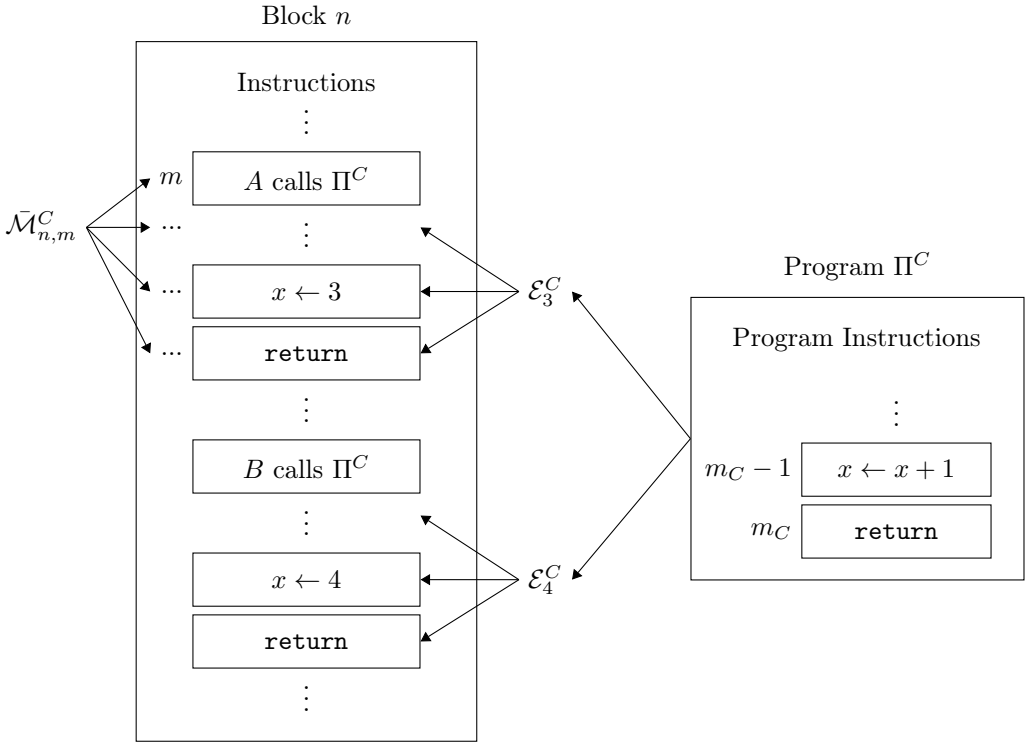


Figure 8: Execution contexts with $x \coloneqq 0$ at call $k = 0$ to Π^C.

The definitions utilized here are purposefully general and not necessarily fully consistent with the terminology in any specific blockchain implementation, as these are meant only to provide an abstraction of properties that apply across a range of blockchain implementations. In particular, we have largely ignored transaction costs, storage costs, bytecode, representations of data, execution stacks and block validation mechanisms.

The finer practical details of the above concepts varies **substantially** across different implementations of blockchains (even the concept of a contract is not universal and may only be available in a limited form - or not at all). For a more complete description

of these terms along with a detailed, concrete specification of a blockchain implementation supporting (smart) contracts, see the Ethereum white paper (Buterin, 2013) and Ethereum yellow paper (Wood, 2014).

3.2 Caveat: non-uniqueness of intrablock state

Throughout Section 2 we worked under the assumption of *stochastic processes*: sequences of real-valued random variables indexed by a (totally ordered) time index $t \geq 0$. As general as this may seem, this is apparently **not** satisfied for a blockchain with multiple simultaneous states at the same block time t_n (without some alternative temporal structure imposed on this, as explored in the next sections).

For a given block time $t_n, n \in \mathbb{N}_0$ representing a (physical) point in time $t \geq 0$ of the n'th block, the associated block will generally consist of $m_n \gg 1$ non-empty instructions, e.g. transfers between accounts (Nakamoto, 2009). This then leads to complications in the interpretation of the overall block state, or even specific elements of this, as stochastic processes indexed by the time of the containing block, since there can be multiple mutations of the same element within a single block with only one associated time t_n.[5]

Figure 9 shows an example for an account A where multiple mutations of account balances after asset transfers involving account A in block n leads to ambiguity in referencing this balance "at time t_n" - this value is not unique, as **any** of \$22, \$17, \$19 and \$6 would correspond to the balance at *some* stage in the sequential execution of instructions within the block.

For *smart contract compatible* blockchains in particular, such as Ethereum (Buterin, 2013), the practical ramifications of the order of transactions can be significant, as instructions may themselves be contingent on the state of the blockchain prior to their execution. Concrete examples of this include Automated Market Maker (AMM) contracts (Mohan, 2022), where the exchange rate between two assets is determined by the ratio of these as held by the contract account at the point of the contract call. Since this ratio is mutated on every call to the AMM contract resulting in an exchange, different exchange rates may apply to contract calls *within the same block*.

Unlike the traditional limit order book approach prevalent in "traditional finance" (Gould et al., 2013), the order of transactions in blocks is **not** necessarily determined by their time of submission (in spite of their sequential execution). Since submitted transactions can be observed prior to the finalization of blocks, this can lead to potential single-block frontrunning (Daian et al., 2020).[6] This is related to the more general notion of Maximal Extractable Value (MEV), which denotes the excess value extractable by controlling the inclusion-, exclusion- and order of transactions within a given block by *block validators*

[5]The state of the block with time t_n will itself **not** generally be $\mathcal{F}_{t_n}$-measurable "in real time" due to the delay from validation of blocks - but will be *at least* $\mathcal{F}_s$-measurable for some $s \in [t_n, t_{n+1})$ due to validation of subsequent blocks being dependent on the *finality* (completed validation) of previous blocks.

[6]This would be much like a broker having the ability to rearrange the timing of their trades after observing the realized price evolution at the end of a trading day. Such practices would generally be regarded as fraud in traditional finance.

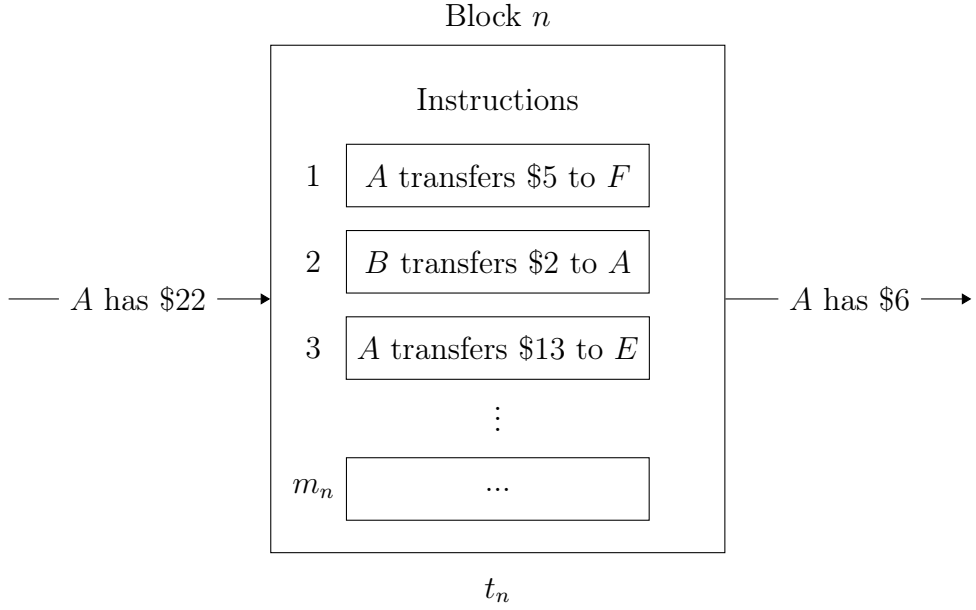


Figure 9: Non-uniqueness of account A balance at time t_n.

(Ethereum Development Documentation, 2024).[7]

Another category of strategies (exploits) related to this peculiar nature of multiple "simultaneous" mutations arises from the concept of flash loans: assets borrowed "early" in a block and paid back in a "later" transaction *in the same block*, enabling the manipulation of certain vulnerable contracts by utilizing this transient capital in the mutation of intrablock state, all the while having **no** exposure between blocks. Applications of such loans in the manipulation of AMMs and price oracles are covered by Qin et al. (2021).

Worthy of note is that the staking pool implementation presented in the paper of Batog et al. (2018) (see Section 4.2) would be especially vulnerable to manipulation of transaction ordering, since total claims in this setup are determined by an explicit deposit of "rewards", which are allocated *instantly* at the point of the deposit in the block. Since the p shares in this setup are themselves based on accounts *staking* their assets (temporarily transfering custody of these to the contract account), this **heavily** incentivizes manipulation of transaction order in blocks.

Specifically, an account may attempt to place their staking contract call just before a large deposit of rewards (increasing their share of these), followed by immediately unstaking those same assets - all within the same block. Taking this a step further, this relative share could be inflated by also placing any other staking transactions in the block *after* the distribution of rewards (and all unstaking transactions *prior* to it). This may defeat the very purpose of such contracts: namely, incentivizing locking the assets in question *over time* to accrue claims, akin to dividend payments of stocks.

Fortunately, the restriction to a single update of ξ per (block) time already implemented in Algorithm 3 mitigates such issues to a large extent. In particular, the above mentioned example of an account increasing their shares immediately before a deposit would **not**

[7]The extent to which this excess value implies inefficiencies in practice will vary **wildly** across blockchains. In the case of Ethereum, this may translate into higher transaction costs paid out to validators, as transacting accounts outbid each other to achieve more favorable outcomes in the block.

work, since accumulation of claims requires at least one additional evaluation of ξ at a later block - implying both that the total shares would still "have time" to increase in the remainder of the block regardless of their order **and** that no claims can be allocated to new shares within the same block. This significantly reduces the incentives to "bribe" validators to induce a specific transaction order (which would ultimately push transaction costs up as accounts outbid each other for favorable orders of transactions, with no net gain in total claims).

Beyond the potential inefficiencies discussed above, the ambiguity in referencing on-chain state "at time t_n" leads to difficulties in translating *any* set of methods that assumes a temporal flow of state and information, as in Section 2. This motivates the formulation of notions that enable an interpretation of these sequences of block state as stochastic processes in the context of a probability space equipped with suitable filtrations that model the availability of block state information to contracts. The next section proposes one such interpretation, along with an on-chain monitoring approach for cumulative deposits that enables the implementation of the special cases covered in Section 2.3 in light of the limited information available to contracts based on the introduced filtrations.

3.3 Block process, filtrations and monitoring

3.3.1 Block state as random variables

We assume a probability space $(\Omega, \mathcal{F}, P)$ representing some probabilistic structure of the blockchain and a measurable space (S, Σ), consisting of the set S of all possible block states and a σ-algebra Σ on S.[8]

Denote by $B_{n,m}$ with $n, m \in \mathbb{N}_0$ the (random) block state of the n'th block after the m'th instruction, taking on some value in S. In order to characterize these random variables as a stochastic process, it remains to find some totally ordered index that provides a chronological ordering of these indices to arive at an indexed collection $B \coloneqq (B_t)_{t \geq 0}$ of random variables. As discussed in Section 3.2, utilizing the block timestamps t_n does **not** accomplish this, as multiple mutations may be contained within a single block.

One way of resolving non-uniqueness in the indexing of states of a single block is to utilize a two-dimensional index set (n, m) to refer to block $n \in \mathbb{N}_0$ and instruction $m \in \mathbb{N}_0$ with all $m \geq m_n$ mapping to the state after the last (non-empty) instruction in the block. While this provides an ordering among blocks and an ordering of instructions within blocks, it is not sufficient for the definition of a stochastic process, since we still have no *totally ordered* index that induces a chronological order on both: e.g. mapping a point in time $t \geq 0$ to a (single) random variable $B_{n,m}$.

3.3.2 Time index of instructions

Define the *instruction time* as

$$\bar{t}(n, m) \coloneqq n + \sum_{i=1}^{m} \frac{1}{2^m} \tag{3.4}$$

[8] For example, one could let S be the set of all possible configurations of arbitrary-length tuples of bits, with each of these representing some (possibly invalid) state of the blockchain - taking then Σ to be the power set of this choice of S.

for the m'th instruction in block n, mapping the two-dimensional $(n, m) \in \mathbb{N}_0^2$ tuple in Section 3.3.1 to a positive real-valued "time" with

$$\bar{t}(n, m) < \bar{t}(n, m + 1) < \bar{t}(n + 1, m) \tag{3.5}$$

for all $(n, m) \in \mathbb{N}_0^2$, hence preserving the order of blocks and instructions within blocks.

This enables the simultaneous indexing of block- and intrablock state by rounding down "points in time" to the inverse of (3.4), i.e. $\bar{t} : [0, \infty) \to \mathbb{N}_0^2$ with

$$t \mapsto \left(\lfloor t \rfloor, \left\lfloor \frac{-\log(1 + \lfloor t \rfloor - t)}{\log 2} \right\rfloor \right) =: (\bar{n}(t), \bar{m}(t)) \tag{3.6}$$

where $\lfloor x \rfloor$ denotes the (floored) integer part of $x \geq 0$ (Graham et al., 1994). This maps a real-valued $t \geq 0$ to the nearest (rounded down) block index $n \in \mathbb{N}_0$ and instruction index $m \in \mathbb{N}_0$ such that

$$\bar{t}(\bar{n}(t), \bar{m}(t)) =: \widehat{t}(t) \leq t. \tag{3.7}$$

The *block process*

$$B \coloneqq \left(B_{\bar{n}(t), \bar{m}(t)}\right)_{t \geq 0} = \left(B_{\lfloor t \rfloor, \left\lfloor \frac{-\log(1 + \lfloor t \rfloor - t)}{\log 2} \right\rfloor}\right)_{t \geq 0} \tag{3.8}$$

then constitutes a stochastic process indexed by $t \geq 0$.

Imposing this "artificial" temporal structure on the sequence of instructions within- and across blocks then allows for the interpretation of the sequential changes in block state across instructions in a block as happening "over time". One should however be wary of interpreting (3.4) in a literal sense over the physical time t_n associated with the block in any application that relies on time explicitly, such as in the delayed allocation dynamics from Section 2.3.3.[9]

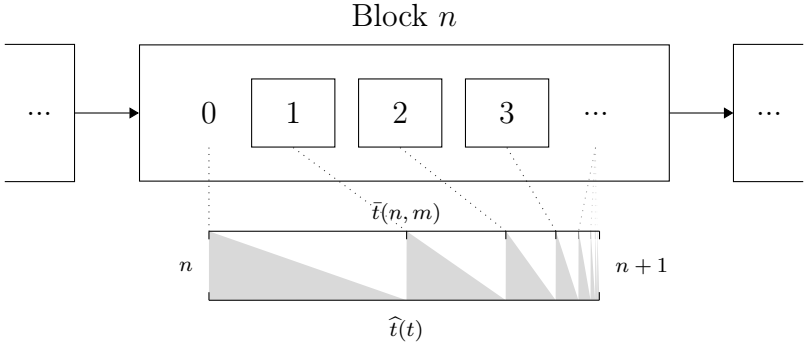


Figure 10: Temporal ordering of intrablock state for $t \in [n, n + 1)$.

With the mapping between times $t \geq 0$ and tuples $(n, m) \in \mathbb{N}_0^2$ already established, we can use time t and tuples of block- and instruction indices (n, m) interchangably, as these

[9]Not only would it be misleading (to put it mildly) to take a literal interpretation of block counts corresponding to *linear* time progression and instruction counts to *non-linear* time progression, the resulting time steps per instruction quickly decay, becoming impractically small to represent numerically. The key point is that a totally ordered index that captures the sequential nature of intrablock state changes exists, leading to a valid interpretation of blockchains as stochastic processes.

will simply be different representations of the same underlying order: with t mapping to (n, m) via (3.6) and (n, m) mapping to t via (3.4). For any given time $t \geq 0$, we can retrieve the exact instruction time via $\widehat{t}(t)$, as this "rounds down" arbitrary points in time to the nearest realized instruction time as shown in Figure 10. For simplicity, we take "instruction at time t" as referring to the instruction of the *rounded down* time $\widehat{t}(t)$.

3.3.3 Natural filtration of the block process

While the structure of a stochastic process indexed by $t \geq 0$ via (3.8) is now in place, we have yet to define a suitable *filtration* for this block process, i.e. the characterization of the flow of block state *information* (from block to block and instruction to instruction) accessible to executing instructions.

It may seem counter to the simultaneous nature of instructions in a block to not assume measurability of the whole block at once, and while true from the perspective of physical time, the availability of information to (program) instructions in a block more closely resembles that of a temporal interpretation of the sequence of instructions, i.e. the m'th instruction of block n can depend on the state prior to its execution, but does not in general have access to the block state at instructions $k > m$ "in the future".[10]

This leads us to define the *block filtration* $\mathbb{F}^B = (\mathcal{F}_t^B)_{t \geq 0}$ with

$$\mathcal{F}_t^B \coloneqq \sigma\left(B_s^{-1}(E) : 0 \leq s \leq t, E \in \Sigma\right), \tag{3.9}$$

which is the *natural filtration* consisting of sub-σ-algebras generated by the block process (3.8) over time, making B an $\mathbb{F}^B$-adapted process by construction. This models the history of changes in block state after the execution of instructions up to (and including) the latest instruction at time t - but **not** future instructions, even those within the same block.

3.3.4 Contract filtrations

While the block filtration (3.9) characterizes the changes in state across instructions, of greater practical relevance in the implementation of contract programs is the information generally available in the execution contexts of a contract program Π^C.

Typically, the full history of past states will **not** be accessible to contracts at the time of their execution, but rather only the state of the block *before* the instruction is executed, as in the definition of $\mathcal{M}_{n,m}^C$ (3.1). Elements of this state (such as the balance of an account) can however be stored within the contract storage for future reference. In this sense, all *observable* information by program instructions can be made available at all points in the future (ignoring any contract storage limitations) - but unobserved past states remain unobserved, consistent with the definition of a filtration as an indexed family of sub-σ-algebras.[11]

[10] If this **was** possible in general, one could create infinitely recursive contingencies by having instruction m depend on the state at instruction $k > m$ and instruction k depend on the state at instruction m.

[11] This will ultimately come down to the specific blockchain implementation, however even if a blockchain *would* provide the capability to look up past states from within a contract, filtering and iterating over unobserved states would generally be prohibitively expensive in a validated "on-chain" setting. This would also produce *dynamic* transaction costs that may not be known until the time of execution, since the number of required iterations might depend on the order of transactions in the block.

Assessing measurability on a per-instruction level is however too granular for anything but the lowest level of interpretation. In particular, B_t is not itself observable by the instruction at time t in general, since this can mutate B_t (and the instruction cannot, at least in a literal sense, access the result of its own execution ahead of time). This would then require arguing for the measurability of an element of B_t at the level of *individual program instructions*. Even a process which is *entirely* determined by program instructions would not necessarily be observable under such an interpretation - making it cumbersome to reason about elements of B being "observable to the contract" in a more general sense.

Based on this remark, we define

$$\mathcal{S}_t^C \coloneqq \left\{ s \geq 0 : \bar{m}(s) \in \bar{\mathcal{M}}^C_{\bar{n}(t),\bar{m}(t)} \right\} \cup \{0\} \tag{3.10}$$

as the set of times of instructions immediately preceding- or being *within* execution contexts that have been reached by time t, along with the initial time 0. In particular, this may include times $s > t$ when t is itself within an execution context of C.

From (3.10), we finally define the *contract filtration* as $\mathbb{F}^C \coloneqq (\mathcal{F}_t^C)_{t\geq 0}$ with

$$\mathcal{F}_t^C \coloneqq \sigma\left(B_s^{-1}(E) : s \in \mathcal{S}_t^C, E \in \Sigma \right), \tag{3.11}$$

which models the history of all block state available within execution contexts of contract C reached by time t. By this definition, $\mathcal{F}_t^C$ is still defined for all $t \geq 0$, but remains constant between execution contexts, "jumping" from the information available at the latest execution directly to **all** the information (i.e. block states) available within any execution context reached by time t.

The rationale for including "future" times $s > t$ of on-going execution contexts is that programs are themselves a deterministic sequence of instructions: any mutations happening within execution contexts will themselves be predetermined, since the *random* block state will have been realized prior to the contract call. In a sense, a contract program whose instructions can observe an element *somewhere* in its execution can be augmented to observe it **anywhere** in its program by replicating instructions appropriately, so we can skip the extra work of assessing measurability at the level of individual instructions - though we must then avoid interpreting $\mathcal{F}_t^C$-measurability as an element being *accessible* at time t, but rather that it can be *made* accessible.

The contract filtration (3.11) is especially useful in assessing the reliance on processes which cannot always be observed between execution contexts. In particular, the block process B as a whole **cannot** be $\mathbb{F}^C$-adapted for **any** contract C: for any $s \leq t$ with $s \notin \mathcal{S}_t^C$, the contract filtration will be constant, and hence B_s will not be $\mathcal{F}_s^C$-measurable.

B_s **will** however be $\mathcal{F}_s^C$-measurable for $s \in \mathcal{S}_t^C$: the contract can access any block state within its own execution context. Unlike elements of the block filtration (3.9), $\mathcal{F}_t^C$ will not contain the block states from past times $s \notin \mathcal{S}_t^C$ outside its execution contexts: unobserved "gaps" remain unobservable in the future. Regardless, $\mathbb{F}^C$ is not strictly smaller than $\mathbb{F}^B$, since $\mathcal{F}_t^C$ may contain block states B_s for $s > t$ when t is within an execution context of C, and these block states would **not** be in $\mathcal{F}_t^B$ until *after* the termination of the program.

Processes which can change only from within execution contexts (e.g. those represented

by elements of contract storage) will always be $\mathbb{F}^C$-adapted, since all times where these could change will be in $\mathcal{S}_t$ by definition. As alluded to previously, this would **not** be the case if taking the filtration to be generated only from the times of directly observable block state - this property is a direct result of treating all block state information as "measurable" at the *start* of execution contexts.

The contract filtration provides a useful characterization of the information that can accessed from within the *overall* execution of a contract C, and we take filtrations $\mathbb{F} := \mathbb{F}^C$ of this form in the interpretation of Section 2. This still implies some additional work, since this assumes a fixed choice of contract C: and we have implicitly relied on processes which **cannot** be $\mathbb{F}^C$-adapted in general, namely the cumulative deposit processes in Section 2.3. The subsequent section shows how a contract C can be amended with appropriate variables and program instructions to ensure that an $\mathbb{F}^C$-adapted *monitored* representation of cumulative deposits can still be observed, providing a lower bound that can be utilized in delayed claim allocation dynamics between execution contexts.

3.3.5 Monitoring of cumulative deposits

We now utilize the notions covered in previous sections to establish a *monitoring* approach that achieves $\mathbb{F}^C$-adapted lower bounds (for some suitable contract C) on the cumulative deposit process Y^+ relied on in Section 2.3. Starting from some arbitrary contract C with a program Π^C, we iteratively amend this program to achieve the desired properties. To start with, we focus on the fully backed case from Section 2.3.2 and assume the claims account is the C contract account.

First note that deposits into the claims account will not be limited to program instructions in general, as (non-contract) accounts may freely transfer into this account. This already shows that Y^+ is **not** $\mathbb{F}^C$-adapted, since changes can occur *between* execution contexts of C (where $\mathbb{F}^C$ remains constant).

Furthermore, even at times $s \in \mathcal{S}_t^C$, we will not be able to directly observe the cumulative withdrawals Y^- from the isolated block state B_s: we can only observe the *current* account balance Y_s. A persistent variable representing Y^- in contract storage is thus required to track this quantity.

Unlike deposits, withdrawals (redemption of claims) **always** happen in the execution context of C, as the program maintains exclusive control over the contract account. It is then straightforward to keep **exact** track of Y^- by incrementing the relevant variable in contract storage by the withdrawn (redeemed) amount after every such instruction, thereby making Y^- an $\mathbb{F}^C$-adapted process as captured by this variable.

By definition (2.25), we have the identity

$$Y_t^+ = Y_t + Y_t^-, \tag{3.12}$$

and since Y^- is $\mathbb{F}^C$-adapted by the argument above and Y can be observed directly from the block state at **any** time $s \in \mathcal{S}_t^C$, this identity allows us to infer Y^+ at all times $s \in \mathcal{S}_t^C$. However, (3.12) is **still** not $\mathbb{F}^C$-adapted, since Y is not $\mathcal{F}_s^C$-measurable in general for $s \leq t$ when $s \notin \mathcal{S}_t^C$.

We therefore take the *monitoring time* at time $t \geq 0$ to be

$$t^C := t \wedge \max\left(\mathcal{S}_t^C\right), \tag{3.13}$$

which will simply be t whenever $t \in \mathcal{S}_t^C$ or the last time prior to t within the execution context of C otherwise, hence $t^C \in \mathcal{S}_t^C$ for all $t \geq 0$.

From this we finally define the *monitored cumulative deposits* as the process

$$Y_t^{+C} := Y_{t^C} + Y_{t^C}^- \overset{\text{a.s.}}{=} Y_{t^C}^+ \tag{3.14}$$

which **is** $\mathbb{F}^C$-adapted, since $t^C \in \mathcal{S}_t^C$ ensures that this will be $\mathcal{F}_t^C$-measurable.

The monitored process (3.14) corresponds to the **exact** value of Y^+ within execution contexts, and since Y^+ is non-decreasing, this becomes a non-strict lower bound of Y^+ between execution contexts, enabling a practical implementation of the methods relying on cumulative deposits in Section 2.3 by utilizing this monitored (lower bound) process Y^{+C} in place of Y^+ in the claim dynamics *between* execution contexts.

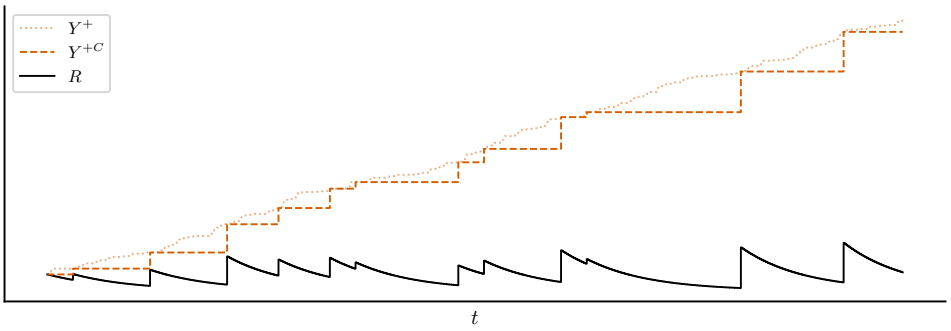


Figure 11: Cumulative deposits Y^+, monitored cumulative deposits Y^{+C} and remaining deposits process R.

We next consider the partially backed variant in Section 2.3.4, where the separation of the oustanding balance L assumes the ability to deposit to- and withdraw from the claims account with no impact on Y^- and Y^+. Contracts may accomplish this by incorporating a dedicated deposit / withdrawal mechanism within the contract program, which will increment a variable representing L on withdrawals (**without** also incrementing Y^-) and decrement it on deposits. This will thus track these "excluded" inflows- and outflows from the contract account, ensuring that L is $\mathbb{F}^C$-adapted through its representation by the L variable in contract storage.

This however means that the net balance observed in the monitoring will actually be $Y - L$ (the amount held by the account), hence the *monitored partially backed cumulative deposits* will be

$$Y_t^{+C} := (Y_{t^C} + L_{t^C}) + Y_{t^C}^- \overset{\text{a.s.}}{=} Y_{t^C}^+ \tag{3.15}$$

to correct for the amount L missing in the net balance state. This is thus a generalization of the previous definition (3.15), where $L \overset{\text{a.s.}}{=} 0$ at all times corresponds to the fully backed variation. Otherwise, the monitoring process remains unchanged.

It is important to note that if we were to evaluate the delayed allocation of claims in

Section 2.3.3 *after* adding the *immediately observed* value of the monitored deposits at times $s \in \mathcal{S}_t^C$ to R via the ODE dynamics in (2.31), we would be *overshooting* the allocation dynamics, since this would include **all** deposits in the accrual over time as if they had been present at the start of the interval.

Instead, the evaluation of the ODE dynamics when incrementing ξ for a given block should happen **before** incorporating the increase in monitored cumulative deposits into the remaining rewards process R, similar to how lagged values of p and q are used in Algorithm 3. Otherwise, the value would effectively act as an *upper bound* on Y^+ on the interval in question, rather than a lower bound.

3.3.6 Delayed claims and robustness

We conclude by discussing how the delayed allocation of claims from the monitored deposit process leads to an even greater degree of *robustness* to potential MEV-related frictions than already ensured by the prevention of multiple single-block updates discussed in Section 3.2.

First note that when utilizing the monitored deposits process in the implementation of delayed claims, the incremented value of ξ in any block $n > 0$ can be known independently of *any* instructions in the block besides the initial timestamp t_n. This is because ξ is derived from the (deterministic) ODE dynamics evaluated from the (lagged) monitored deposits, which is itself determined from monitoring in *prior* blocks. While the variables representing the monitored process may be updated after increments of ξ within the same block, these will have no impact on ξ until later blocks.

This invariance to the order of transactions that increment ξ is **not** guaranteed in general, even when preventing multiple updates in the same block, as the total claims X may itself be non-constant within a block. In fact, the explicit prevention of multiple updates in a single block is no longer necessary, as the continuous ODE dynamics (2.31) already **guarantee** that any increments to ξ after the first in a block will **always** be zero when no additional time has elapsed, even when Y^+ increases in the same block (since the immediate jumps in Y^+ cancel).

Delaying claims further disincentivizes adversarial behavior related to manipulation of blocks more generally (such as excluding transactions that increase the shares of other accounts): by "smoothing out" the allocation of claims over time (i.e. multiple blocks), the potential short-term gains of even outright exclusion of other transactions within a single block are dampened, as the net gain in allocated claims will only apply to the claims of that single isolated block. Sustaining such highly-targetted exclusion of transactions over *consecutive* blocks will be economically **infeasible** on any "healthy" blockchain, severely limiting the potential upside of such tactics in contrast to implementations where claims are allocated in (large) single-block bursts.

The delayed claims from the monitored process thus enables a high degree of resistance to manipulation of blocks by aligning the incentives of accounts: **all** accounts with positive relative shares benefit from collectively monitoring new deposits, as these can only increase the total allocation of claims. Meanwhile, internal (zero-sum) competition among accounts to manipulate blocks by reordering- or excluding transactions at the expense of other accounts within a given block is limited for the reasons given above.

This concludes the theoretical aspects of the article, with the subsequent section providing pseudocode for concrete implementations of contracts, showing how the theoretical methods of Section 2 can be translated into practice based on the notions introduced throughout the preceding sections.

4 Contract implementations

4.1 Staking pools

Staking pools are a special case of claim distribution contracts where shares p^A correspond one-to-one to assets *staked* by accounts (transferred temporarily to the contract account until *unstaked*). Relative shares thus correspond to the quantity of assets staked by an account in proportion to the total quantity of assets staked across all accounts.

These contracts have several use cases, for example as an analogy to dividend-paying stocks, with the *staking asset* being some representation of ownership and *claims* representing dividend payments deriving from revenue generated by the underlying business. Such contracts can be further embedded with e.g. voting capabilities based on the relative stakes of accounts.

Typically these contracts will not utilize the same asset for both staking and paying out claims; often the staking asset is some (illiquid) token representing ownership in a project, while the *claimable asset* would be a highly-liquid asset (e.g. the native token of the blockchain or a stablecoin). For simplicity, we leave explicit references to assets out of the implementations covered here and simply take these to be implicit in the `transfer` instructions.

Both of the subsequent examples assume the fully backed setup covered in Section 2.3.2, and hence will rely on the monitored process $Y^+ = Y + Y^-$, where Y is the current contract account balance (of the claimable asset) and Y^- is a global variable tracked in the contract as described in Section 3.3.5.

4.2 Instant allocation of claims

Contract 1 shows the simplest example of a fully backed staking pool contract where claims are allocated instantly, i.e. $X := Y^+$, such that total claims are based on the *immediate* allocation of **all** cumulative deposits. This mirrors the method presented in Batog et al. (2018), though with the added properties of dynamically monitored deposits (instead of these only being registered on explicit deposit contract calls) as outlined in Section 3.3.5, as well as the prevention of multiple increments to ξ at a single point in time (i.e. within the same block) as in Algorithm 3 and Section 3.2.

Contract 1 Distributor contract with instant allocation of deposits.

Globals: $\xi, Y_-^+, Y^-, q, t_- \leftarrow 0$ ▷ *Initialize global variables*
Locals: $\xi^A, X^A, p^A \leftarrow 0$ ▷ *Initialize local (per account) variables*
▷ *Increment account A claims* ◁
function _INCREMENT_CLAIMS(A)
    if $q > 0 \wedge$ `now()` $> t_-$ **then**
        $\xi \leftarrow \xi + \frac{1}{q}\left(Y^+ - Y_-^+\right)$ ▷ *Increment ξ based on change in deposits*
        $Y_-^+ \leftarrow Y^+$ ▷ *Store for next change in $X = Y^+$*
    $X^A \leftarrow X^A + p^A\left(\xi - \xi^A\right)$ ▷ *Increment X^A account claims*
    $\xi^A \leftarrow \xi$ ▷ *Store ξ for next increment of account claims*
    $t_- \leftarrow$ `now()` ▷ *Store current time to prevent simultaneous updates*
▷ *Stake assets for account A* ◁
function STAKE(A, Δp)
    A `transfer` Δp `to` C ▷ *Transfer from A to contract*
    _INCREMENT_CLAIMS(A) ▷ *Increment X^A prior to updating share*
    $p^A \leftarrow p^A + \Delta p$ ▷ *Update account A share*
    $q \leftarrow q + \Delta p$ ▷ *Update total shares*
    return
▷ *Unstake assets for account A* ◁
function UNSTAKE(A, $-\Delta p$)
    `assert` $p^A \geq \Delta p$ ▷ *Ensure new share will be non-negative*
    C `transfer` Δp `to` A ▷ *Transfer from contract to A*
    _INCREMENT_CLAIMS(A) ▷ *Increment X^A prior to updating share*
    $p^A \leftarrow p^A - \Delta p$ ▷ *Update account A share*
    $q \leftarrow q - \Delta p$ ▷ *Update total shares*
    return
▷ *Redeem claims for account A* ◁
function REDEEM(A)
    _INCREMENT_CLAIMS(A) ▷ *Increment X^A prior to redemption*
    C `transfer` X^A `to` A ▷ *Transfer from contract to A*
    $Y^- \leftarrow Y^- + X^A$ ▷ *Increment cumulative withdrawals*
    $X^A \leftarrow 0$ ▷ *Reset X^A after redemption*
    return

4.3 Delayed exponential allocation

Contract 2 shows an example of *delayed claim allocation* as covered in Section 2.3.3, with an exponential decay of remaining rewards based on some rate $r > 0$. A global variable R representing remaining rewards is added in this setup, and as discussed in Section 3.3.5, new deposits are incorporated into the remaining rewards *after* evaluating the dynamics. Another global variable Y_-^+ is utilized to track the changes in cumulative deposits between evaluations of ODE dynamics. Otherwise, the STAKE, UNSTAKE and REDEEM functions are all taken to be identical to those of Contract 1.

Multiple increments of ξ in a single block are implicitly suppresed by the dynamics, hence the t_- variable is moved into the increment function itself such that this represents the time of the last *actual* evaluation of claims (where $q > 0$) - though one could also utilize the previous setup to "pause" the dynamics in periods with $q = 0$, which would then

remove the jump term in X^Q that had to be explicitly dealt with in Section 2.2.

Besides the added robustness to MEV attacks from delayed allocation of claims in general, the exponential specification in particular has some additional nice properties. For example, the deposits will never "run out", allowing for continuous distribution of claims *indefinitely* with no sudden termination of allocations.

Additionally, the *rate* of exponential decay has an interpretation independent of the units of the claimable asset and the scale of cumulative deposits. Exponential growth of revenue is one example of a common economic phenomenon that demonstrates why this is a useful property. In contrast, utilizing a *linear* rate of claim allocation in revenue sharing- and dividend contexts would lead to ill-suited behavior if revenue grows (or shrinks) over time. Regardless of this, the latter is **widely** used in practice.

Computing the exponential function in a smart contract context is a non-trivial task on its own however, as blockchains generally only provide (unsigned) integers and no built-in capabilities for non-arithmetic math operations. Custom implementations of approximation algorithms utilizing *fixed-point numbers* are thus typically required (see Auster, 2024).

Contract 2 Distributor contract with delayed exponential allocation of claims.

Globals: $\xi, X_-, Y_-^+, Y^-, R, q, t_- \leftarrow 0$ ▷ *Initialize global variables*
Locals: $\xi^A, X^A, p^A \leftarrow 0$ ▷ *Initialize local (per account) variables*
▷ *Increment account A claims* ◁
function _INCREMENT_CLAIMS(A)
    if $q > 0$ **then**
        $R \leftarrow Re^{-r(\texttt{now}()-t_-)}$ ▷ *Evaluate ODE dynamics*
        $R \leftarrow R + (Y^+ - Y_-^+)$ ▷ *Incorporate new deposits after dynamics*
        $\xi \leftarrow \xi + \frac{1}{q}(Y^+ - R - X_-)$ ▷ *Increment ξ based on change in total claims*
        $X_- \leftarrow Y^+ - R$ ▷ *Store for next change in $X = Y^+ - R$*
        $Y_-^+ \leftarrow Y^+$ ▷ *Store for next change in Y^+*
        $t_- \leftarrow \texttt{now}()$ ▷ *Store for next evaluation of ODE dynamics*
    $X^A \leftarrow X^A + p^A\left(\xi - \xi^A\right)$ ▷ *Increment X^A account claims*
    $\xi^A \leftarrow \xi$ ▷ *Store ξ for next increment of account claims*
▷ *Stake assets for account A* ◁
function STAKE(A, Δp)
    ...
▷ *Unstake assets for account A* ◁
function UNSTAKE(A, $-\Delta p$)
    ...
▷ *Redeem claims for account A* ◁
function REDEEM(A)
    ...

5 Conclusion

This article introduced a general stochastic framework for the distribution of claims in continuous time. A scheme was derived allowing for constant time evaluation of account claims in this setup. Practical special cases of the general setup were then discussed. Notions were introduced allowing for an interpretation of blockchains as stochastic processes in the general setup, along with filtrations to model the flow of information available in the execution of smart contracts. The concept of process monitoring was introduced to enable the implementation of claim allocations that derive from cumulative deposits. Concrete pseudocode implementations were then provided.

References

Auster, J. (2024). Approximation of the exponential function with fixed-point numbers. Working Paper. Available at SSRN: http://dx.doi.org/10.2139/ssrn.4850002.

Batog, B., Boca, L., and Johnson, N. (2018). Scalable Reward Distribution on the Ethereum Blockchain. *DappCon Berlin*.

Buterin, V. (2013). Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform.

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., and Juels, A. (2020). Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE.

Ethereum Development Documentation (2024). Maximal Extractable Value (MEV). https://web.archive.org/web/20240702125613/https://ethereum.org/en/developers/docs/mev/. Archived: July 2, 2024.

Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., and Howison, S. D. (2013). Limit order books. *Quantitative Finance*, 13(11):1709–1742.

Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete mathematics*. Addison Wesley, Boston, MA, 2 edition.

Kloeden, P. E. and Platen, E. (1992). *Numerical Solution of Stochastic Differential Equations (Stochastic Modelling and Applied Probability)*. Springer.

Knuth, D. E. (1997). *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., USA.

Mohan, V. (2022). Automated market makers and decentralized exchanges: a DeFi primer. *Financial Innovation*, 8(1).

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.

Nikeghbali, A. (2006). An essay on the general theory of stochastic processes. *Probability Surveys*, 3:345–412.

Qin, K., Zhou, L., Livshits, B., and Gervais, A. (2021). Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In *Lecture Notes in Computer Science*, page 3–32. Springer Berlin Heidelberg.

Revuz, D. and Yor, M. (1999). *Continuous Martingales and Brownian Motion*. Springer Berlin Heidelberg.

Wood, G. (2014). Ethereum: a secure decentralised generalised transaction ledger.

A cacheable two-step method for equitable integer allocation under constraints

Johan Auster*

Abstract

This article outlines a method for solving optimization problems pertaining to "evenly" distributing a budget of integers under constraints on individual allocations. We first show how the real-valued version of the problem can be reformulated to a single-variable problem with a monotonic piecewise linear characterization of solutions. A method for efficiently constructing a table of these solutions by traversing the bounds in sorted order is then demonstrated, which allows for quickly retrieving characterizations of solutions via lookups on this table in a first step. This characterization is then transformed into integer allocations that solve the original optimization problem in a second step. Applications of the method to the dynamic allocation of screen real estate in graphical user interfaces is then demonstrated.

Keywords: Optimization, integer programming, budget allocation.

*Department of Mathematical Sciences, University of Copenhagen, Denmark. E-mail: johan.auster@math.ku.dk.

1 Introduction

Consider an integer $B \in \mathbb{Z}$ representing a budget of some kind. We want to **fully** allocate this budget "as evenly as possible" among $N \in \mathbb{N}$ allocations $A \coloneqq (a_1, a_2, \ldots, a_N)$. More precisely, we want to find an N-tuple of real-valued allocations A^* that is *optimal* in the sense of minimizing the sum of squared residuals from its mean, i.e. minimizing:

$$f(A) \coloneqq \sum_{n=1}^{N} \left(a_n - \frac{\sum_{m=1}^{N} a_m}{N} \right)^2 = \sum_{n=1}^{N} \left(a_n - \frac{B}{N} \right)^2, \tag{1.1}$$

with the latter equality following from the requirement to fully exhaust the budget. With no constraints and no requirement for the allocations themselves to be integer, the solution is then to simply take $a_n \coloneqq {}^{B}\!/_{N}$ for all $n \in \{1, 2, \ldots, N\}$ such that each allocation corresponds exactly to the mean allocation.

If we require the allocations to also be integer, solutions are no longer necessarily unique; for example, given a budget $B = 4$ with $N = 3$, the allocations

$$A \in \{(1, 1, 2), (1, 2, 1), (2, 1, 1)\} \tag{1.2}$$

are all valid solutions with $f(A) = \frac{1+1+4}{9} = {}^{2}\!/_{3}$, while e.g.

$$A \coloneqq (0, 1, 3) \tag{1.3}$$

is not, since then $f(A) = \frac{16+1+25}{9} = {}^{14}\!/_{3} > {}^{2}\!/_{3}$.

We extend this setting further by allowing for *constraints* on individual allocations a_n, requiring each of these to fall within some range of allowed values. The general form of the integer allocation problem can then be fully stated in standard form (Boyd and Vandenberghe, 2004) as finding optimal allocations $A^* \in \mathbb{Z}^N$ such that

$$A^* = \underset{A \in \mathbb{Z}^N}{\arg\min}\, f(A) \tag{1.4}$$

subject to $\sum_{n=1}^{N} a_n = B$ and

$$\begin{aligned} L_1 \leq & a_1 \leq U_1, \\ L_2 \leq & a_2 \leq U_2, \\ & \ldots, \\ L_N \leq & a_N \leq U_N \end{aligned} \tag{1.5}$$

with $-\infty \leq L_n \leq U_n \leq \infty$, $L_n < \infty$, $U_n > -\infty$ and a_n integer for all $n \in \{1, 2, \ldots, N\}$.

While it is possible for multiple solutions to exist as in (1.2), it may also be the case that *no* solutions exist. In particular, if all upper bounds are finite, we cannot fully exhaust a budget $B > \sum_{n=1}^{N} U_n$ without violating *at least* one constraint, and vice versa for lower bounds. For the remainder of the article, we always assume that one- or more allocations has at least one finite lower- or upper bound (i.e. not all allocations are fully unconstrained).

In the next section, an efficient method for deriving a lookup table characterization of solutions to the formulated problem *without* the integer constraint on allocations is

presented. We then show how integer allocations that solve the optimization problem (1.4) can be retrieved from this characterization in a second step. An application to the allocation of screen real estate is then presented.

A `Python` implementation of the method introduced in this paper is available on `GitHub` at `https://github.com/austerj/equitable-integers`, including the reproduction of all plots and a suite of tests demonstrating the correctness of the generated output. These tests include comparison against general trust-region constrained methods (Byrd et al., 1999) as implemented in the `SciPy` package (Virtanen et al., 2020) across a large number of test cases.

2 Lookup table of real solutions

We first consider the optimization problem formulated in (1.4) *without* requiring allocations to be integer. Solutions to this (non-linear) optimization problem *could* be found with iterative numerical solvers, e.g. via previously mentioned trust-region constrained methods (Byrd et al., 1999). However, such iterative solvers produce only *approximate* solutions and can be computationally expensive - especially for relatively simple, analytically tractable problems with more direct solution methods available (such as the class of problems under consideration).

With some inspiration from the "trivial" solution of allocating the mean in the unconstrained case, we consider an alternative problem of finding a real-valued $x^* \geq 0$ with

$$a_n \coloneqq \min(\max(x^*, L_n), U_n) \tag{2.1}$$

for all $n \in \{1, 2, \dots, N\}$ that fully exhausts the budget B. This formulation imposes that all allocations are equal to their lower bound, upper bound or $x^* \geq 0$, with the *only* decision variable then being the value of x^* identical across *all* allocations.

The following argument shows that finding an x^* satisfying the single-variable formulation (2.1) is equivalent to solving the general (non-integer) case (1.4) where each allocation is taken as a separate decision variable: assume allocations A exists such that the budget B is fully exhausted without violating any constraints. Then for any two allocations a_n, a_m with

$$a_n < a_m \wedge a_n < U_n \wedge a_m > L_m, \tag{2.2}$$

the sum of squared residuals (1.1) can be reduced by increasing a_n and lowering a_m by the same amount (such that the budget remains fully exhausted) until these are either equal to each other or their upper- or lower bounds respectively. Since a_n and a_m were arbitrary, this holds until *all* allocations are equal if not constrained by their lower- or upper bounds. This also shows that real-valued solutions, when they exist, are *unique*.

The total budget allocated in this single-variable formulation

$$h(x) = \sum_{n=1}^{N} a_n = \sum_{n=1}^{N} \min(\max(x, L_n), U_n) \tag{2.3}$$

is a non-decreasing continuous piecewise linear function between the smallest lower bound and the largest upper bound, with a rate of change corresponding to the number of allo-

cations without a binding constraint at any given point. Figure 1 shows the relationship between h and bounds for an example with $N = 5$ and constraints:

$$\begin{aligned} 1 \leq & a_1 \leq 5, \\ 2 \leq & a_2 \leq 4, \\ -\infty \leq & a_3 \leq 3, \\ 7 \leq & a_4 \leq 10, \\ 9 \leq & a_5 \leq 12. \end{aligned} \tag{2.4}$$

Since h constitutes a surjective mapping from values of x to budgets B, we can directly infer optimal allocations via (2.1) by evaluating a right inverse $g(B) =: x^*$ of h (2.3) such that the allocations $a_n = \min(\max(x^*, L_n), U_n)$ for $n = 1, 2, \ldots, N$ fully exhaust any budget $B \in \mathbb{Z}$ with

$$\sum_{n=1}^{N} L_n \leq B \leq \sum_{n=1}^{N} U_n. \tag{2.5}$$

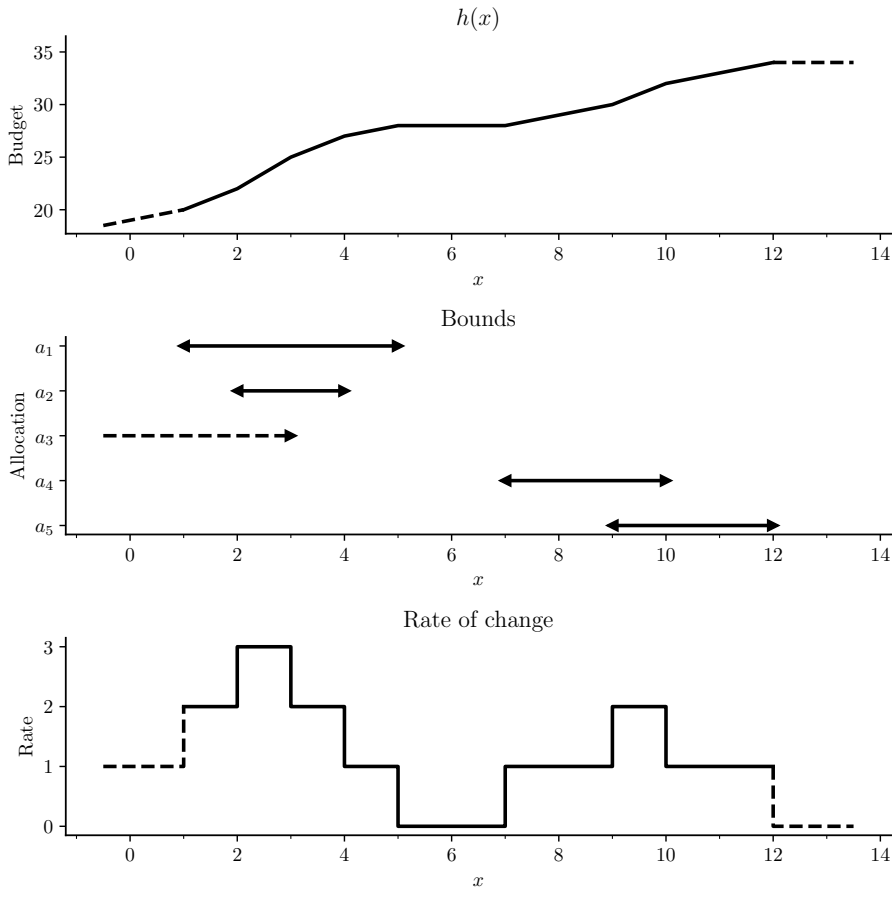


Figure 1: Relationship between the total budget allocated $h(x)$ and the bounds imposed by constraints on individual allocations. Dashed lines represent extrapolation.

A "brute-force" approach to constructing g would be to evaluate h at every unique bound value and infer the linear relationship between each segment. This would re-

quire evaluating $\min(\max(x, L_n), U_n)$ for all N constraints at up to $2N$ points (one for each unique lower- and upper bound value), i.e. upwards of $2N^2$ total evaluations of $\min(\max(x, L_n), U_n)$.

We can achieve the same result by deriving the parameters of the piecewise linear segments directly from the bounds. First we "flatten" the set of lower- and upper bounds into a single tuple of 2-tuples (x_n, F_n) consisting of the value x_n of each bound and a flag F_n denoting whether the value represents a lower bound L or an upper bound U. Doing this for the example in (2.4) would yield the tuple

$$\Big(\big(1, L\big), \big(5, U\big), \big(2, L\big), \big(4, U\big), \big(-\infty, L\big), \big(3, U\big), \big(7, L\big), \big(10, U\big), \big(9, L\big), \big(12, U\big)\Big). \quad (2.6)$$

We then sort this tuple (of tuples), e.g. via Powersort (Munro and Wild, 2018), in increasing order of the bound values x_n after dropping unbounded entries, storing the counts of these as $N_{-\infty}$ and N_{∞}. In the case of (2.6), this would give us

$$\Big(\big(1, L\big), \big(2, L\big), \big(3, U\big), \big(4, U\big), \big(5, U\big), \big(7, L\big), \big(9, L\big), \big(10, U\big), \big(12, U\big)\Big). \quad (2.7)$$

with $N_{-\infty} = 1$ and $N_{\infty} = 0$.

Next, we construct a mapping from values of x to the rates of change in the allocated budget h by iterating over every (x_n, F_n) pair in the sorted tuple (2.7) with the following logic and initial variables $B_- \coloneqq 0$, $r \coloneqq N_{-\infty}$ and an empty hash table $\mathcal{R} \coloneqq \{\}$:

1. Add 1 to r if $F_n = L$ (lower bound), else subtract 1 from r.
2. Set $\mathcal{R}[x_n] \coloneqq r$.
3. Add x_n to B_- if $F_n = L$ (lower bound).

The resulting $\mathcal{R}$ will map values of x to the rates of change in h applicable between each value of x, while B_- will be the sum of all finite lower bounds (or zero if none exist).

Applying this to the example in (2.7) would produce the mapping $\mathcal{R}$:

x	1	2	3	4	5	7	9	10	12
r	2	3	2	1	0	1	2	1	0

Table 1: $\mathcal{R}$ mapping values of x to rates of change r in h.

with $B_- = 19$.

This can finally be transformed into a mapping from budgets to pairs of x and rates of change r by iterating over the (x_n, r_n) key-value pairs of $\mathcal{R}$ (in ascending order of x_n) with the following logic and initial variables $x_- \coloneqq 0$, $r_- \coloneqq N_{-\infty}$, an empty hash table $\mathcal{X} \coloneqq \{\}$ and B_- from above:

1. Add $(x_n - x_-) \cdot r_-$ to B_-.

2. Set $\mathcal{X}[B_-] \coloneqq (x_n, r_n)$.
3. Set $x_- \coloneqq x_n$ and $r_- \coloneqq r_n$.

Continuing with the example from Table 1, we then get:

B_-	20	22	25	27	28	30	32	34
x	1	2	3	4	7	9	10	12
r	2	3	2	1	1	2	1	0

Table 2: $\mathcal{X}$ mapping budgets B_- to values x and rates r.

and from the resulting table, we can retrieve

$$x^* = g(B) = \begin{cases} x + \frac{B - B_-}{r}, & r > 0 \\ x, & r = 0 \end{cases} \tag{2.8}$$

for any budget B on the solution space with x and r retrieved from $\mathcal{X}$:

1. If B is *below* the minimal B_- in $\mathcal{X}$: evaluate (2.8) with the minimal entry values in $\mathcal{X}$ for B_- and x with $r \coloneqq N_{-\infty}$ (extrapolating to the *left*); if $N_{-\infty} = 0$, no solution exists.
2. If B is *above* the maximal B_- in $\mathcal{X}$: evaluate (2.8) with the maximal entry values in $\mathcal{X}$ for B_- and x with $r \coloneqq N_{\infty}$ (extrapolating to the *right*); if $N_{\infty} = 0$, no solution exists.
3. Otherwise, identify the largest B_- in $\mathcal{X}$ that is less than or equal to B, for example via binary search (Knuth, 1998), and evaluate (2.8) with the corresponding entry values for B_-, x and r (interpolating *between* linear segments).

Applying this to Table 2 with e.g. $B = 24$, we get $B_- = 22$ and $x^* = 2 + \frac{24-22}{3} = 8/3$, leading to the allocations

$$A^* = (8/3, 8/3, 8/3, 7, 9)\,. \tag{2.9}$$

A visual representation of the piecewise linear relationship characterized by Table 2 between budgets and values of x^* is shown in Figure 2. Since the concrete example has at least one allocation a_3 without a lower bound, a solution exists for any sufficiently small B, as a_3 can always be lowered to ensure that all other upper bounds can be satisfied by the budget. All allocations have an upper-bound however, hence there exists no solution for $B > 34$, as these would conflict with *at least* one upper bound.

We have now established a methodology for efficiently identifying real-valued solutions via a lookup table constructed from the bounds of the problem. This table can be *cached* (i.e. stored in memory) after construction, allowing for quickly retrieving solutions with the same constraints for a changing budget. In the next section, we show how these solutions can be transformed into (possibly non-unique) integer-valued solutions while satisfying all individual allocation constraints, fully exhausting the budget and maintaining optimality in the sense of minimizing (1.1).

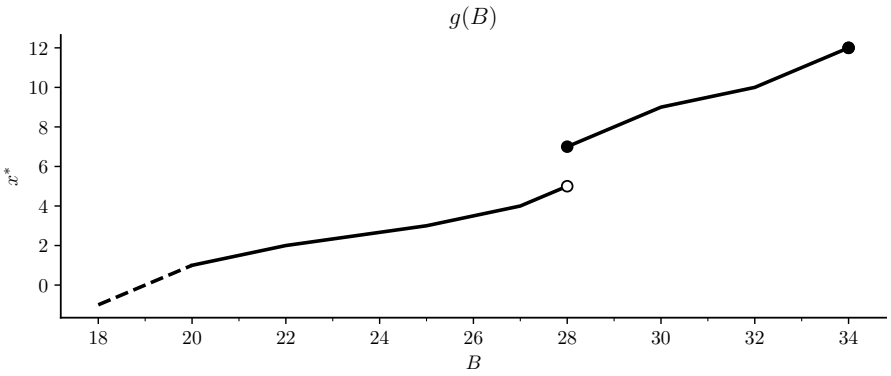


Figure 2: Right inverse $g(B)$ of $h(x)$, mapping budgets B to values x^* that fully exhaust B while satisfying all constraints. Dashed line represents extrapolation.

3 Optimal integer solutions

Although we did not explicitly impose any integer requirement on the bounds in Section 2, we can continue under the assumption that the bounds are themselves integer, since any non-integer bound enforced on an allocation in combination with an integer requirement effectively translates into a rounding of the real-valued constraint in the appropriate direction (e.g. an upper bound of 5.2 is effectively an integer upper bound of 5). We therefore proceed, with no loss of generality, under the assumption that all (finite) bounds are integer-valued.

While the unique solution retrived from the procedure outlined in Section 2 will always fully exhaust the budget, the resulting allocations will not *in general* be integer. Intuitively, the real-valued allocations represent the "ideal" reference state for any potential integer solution: integer allocations that are close to the real-valued optimal allocations will represent comparatively more equitable allocations, and the *additional* requirement for allocations to be integer can only make these solutions comparatively "worse".

With this in mind, we start from a value of x^* and rate r characterizing real-valued solutions for a budget B retrieved via the lookup table from Section 2. As mentioned, and seen in the example allocations (2.9), the resulting allocations are not in general integer. In light of the above discussion, we can then frame the problem of finding integer allocations that minimize (1.1) as *rounding* these (rational) allocations characterized by (2.8) to integers, while still exhausting the budget. In cases where x^* is already integer, no further steps are thus required - so assume that x^* is *not* integer.

First note that any allocations with binding constraints will already be integer, as these will be equal to their lower- or upper bounds. We therefore focus on the remaining (non-integer) allocations with *non-binding* constraints, **all** of which will be equal to x^* prior to rounding. The count of these unconstrained allocations will then correspond to the rate r for the budget in question.

For each of these r unconstrained allocations, we must decide between rounding *down* via flooring $\lfloor x^* \rfloor$ or *up* via ceiling $\lceil x^* \rceil$ (Knuth, 1997). Since the cost function (1.1) is minimized by the original non-integer allocations, the optimal integer allocations must themselves be one of the two possible nearest integer values to x^*.

Letting $\underline{N}$ denote the number of floored allocations and $\bar{N}$ the number of ceiled alloca-

tions, the requirement to fully exhaust the budget will correspond to

$$\underline{N}\lfloor x^* \rfloor + \bar{N}\lceil x^* \rceil = rx^* \tag{3.1}$$

where $\underline{N} + \bar{N} = r$ such that all non-integer allocations are either rounded down- or up.

Using then that $\bar{N} = r - \underline{N}$, it follows that

$$\underline{N} = r\frac{\lceil x^* \rceil - x^*}{\lceil x^* \rceil - \lfloor x^* \rfloor} = r(\lceil x^* \rceil - x^*) \tag{3.2}$$

with $\lceil x^* \rceil - \lfloor x^* \rfloor = 1$ whenever x^* is non-integer. Importantly, (3.2) will **always** be integer, since r divides x^* (2.8) by construction - which also shows that if a real-valued solution exists, at least one integer solution exists.

Since all r unconstrained allocations are identically equal to x^* prior to rounding, the resulting change in the cost function (1.1) will not depend on *which* allocations are rounded down- or up. Knowing the total number of unconstrained allocations $\underline{N}$ to floor and $\bar{N} = r - \underline{N}$ to ceil, applying these to the unconstrained allocations in any arbitrary order (combined with the original constrained allocations) will yield a solution to the integer allocation problem (1.4). We can then, for example, apply flooring to the first $\underline{N}$ unconstrained allocations and ceiling to the remaining unconstrained allocations.

Notably, the above steps do not depend on having already computed the non-integer allocations; only the x^* and r *characterizing* these. Evaluating the integer allocations thus requires little computational overhead compared to the rational solutions. A solution can e.g. be retrieved by iterating over the constraints (L_n, U_n) with the initial variable $i \coloneqq 0$ and an empty tuple A^* with the following logic:

1. Append L_n to A^* if $L_n \geq x^*$.
2. Else, append U_n to A^* if $U_n \leq x^*$.
3. Else, increment i by 1 and append $\lfloor x^* \rfloor$ to A^* if $i \leq \underline{N}$, otherwise append $\lceil x^* \rceil$.

Applying the above steps to the previous example (2.9) from Section 2 with $B = 24$ then leads to the integer allocations:

$$A^* = (2, 3, 3, 7, 9)\,. \tag{3.3}$$

The combination of the precomputed lookup table covered in Section 2 and the fast subsequent retrieval of allocations via the above allows for solving the integer allocation problem with very limited computation. This renders the method especially suitable for real-time applications where computational resources are restricted or prioritized for other tasks. An example of such a context is the allocation of screen real estate in graphical user interfaces (GUIs), as discussed further in the next section.

4 Application to graphical user interfaces

A computer monitor displays information in a rectangular resolution of square *pixels*; for example, a display with a resolution of 1920×1080 will have 1080 "rows", each containing 1920 "columns" of square pixels for a total of around 2 million pixels.

Modern GUIs need to account for a large variety of available screen real estate when supporting different resolutions and aspect ratios. These GUIs may consist of a mix of *flexible* elements taking up an arbitrary amount of additional space (such as the display of long-form text or cells in a spreadsheet), and *fixed* elements, such as toolbars and menus, which *may* be scalable up to a point - but often times will require some *minimal* fixed amount of space, while not benefiting from *additional* space past a certain point.

The presented optimization method can be applied in this context to allocate the total "budget" of available screen real estate across different elements of a GUI, for example via a tree structure of rows containing columns containing rows etc., each embedded with a solver allocating space according to the constraints of the elements nested within it. The bounds for elements can then be chosen to capture the desired behavior, e.g. choosing an upper bound for elements that should only scale up to a point.

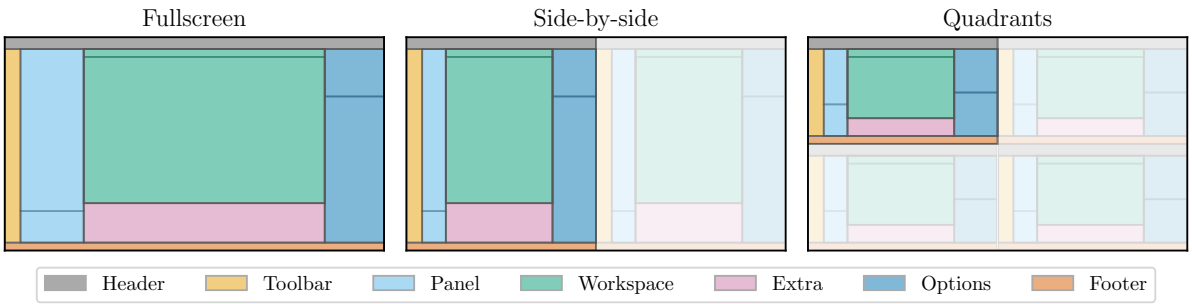


Figure 3: Example of GUI with dynamic allocation of available space.

Figure 3 shows an example of a generic GUI, where the *header*, *toolbar* and *footer* are fixed elements, the *panel* contains some secondary information that can be safely shrunk when insufficient space is available, the *workspace* contains the main content taking up all available space not reserved for other elements, *extra* contains some secondary functionality related to the workspace and *options* contains core functionality related to the workspace, hence always taking up some minimal amount of space. For additional details, see the source code provided in the `GitHub` repository.[1]

Because the proposed method can cache the table characterizing real-valued solutions while requiring minimal computation to retrieve optimal integer allocations for (rapidly) changing budgets afterwards, the allocation of space can be done very efficiently. This enables e.g. the reallocation of space as windows are resized (budgets change) in real-time, while taking only limited resources away from core tasks of an application.

5 Conclusion

This article introduced a two-step method to solving equitable integer allocation problems with constraints. A lookup table characterizing real-valued solutions is constructed in a first step, which can be cached for future lookups with the same constraints for a changing budget. Integer allocations are then retrieved from the lookup table in a second step by applying a precomputed number of flooring- and ceiling operations to the unconstrained allocations. An application to dynamic allocation of monitor real-estate in GUIs was then covered.

[1] `https://github.com/austerj/equitable-integers/blob/main/plots/gui.py`

References

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge, England.

Byrd, R. H., Hribar, M. E., and Nocedal, J. (1999). An Interior Point Algorithm for Large-Scale Nonlinear Programming. *SIAM Journal on Optimization*, 9(4):877–900.

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., USA.

Munro, J. I. and Wild, S. (2018). Nearly-Optimal Mergesorts: Fast, Practical Sorting Methods That Optimally Adapt to Existing Runs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.